



La machine LLM3

J. Chailloux

► To cite this version:

| J. Chailloux. La machine LLM3. RT-0055, INRIA. 1985, pp.54. inria-00070103

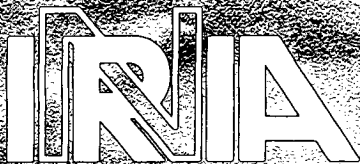
HAL Id: inria-00070103

<https://inria.hal.science/inria-00070103>

Submitted on 19 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél. (3) 954 90 20

Rapports Techniques

N° 55

LA MACHINE LLM3

Jérôme CHAILLOUX

Juin 1985

La machine LLM3

Jérôme Chailloux

Mai 1985

I.N.R.I.A.
Domaine de Voluceau
Rocquencourt
78153 Le Chesnay Cedex
France

Résumé : ce rapport décrit la machine virtuelle LLM3 utilisée pour le transport du système LE LISP de l'INRIA version 15. Il contient également les mesures statiques et dynamiques de l'utilisation des instructions, des opérandes et des accès mémoire de cette machine pour l'écriture du noyau du système LE LISP.

Abstract : this report describes the virtual machine LLM3 used to transport the system LE LISP of INRIA version 15.

1 Introduction

LLM3 est une machine virtuelle conçue pour porter le système LE LISP de l'INRIA [Chailloux 84, 85]. Ce système comprend l'interprète d'un nouveau dialecte du langage LISP [McCarthy 62] appelé LE LISP, son compilateur associé ainsi qu'un important environnement de programmation.

Un transport effectif de la machine LLM3 a déjà été réalisé sur les Unités Centrales suivantes :

- VAX 11 de DEC
- MC68000 de Motorola
- Perkin Elmer 32
- HB68 Multics
- Intel 8086/8088
- IBM série 30xx
- Ridge 32
- SEL 32

Un transport est en cours sur les Unités Centrales suivantes :

- BellMac 32
- DPS7
- PR1 ME
- Mini6

Et en projet sur les Unités Centrales suivantes :

- MAIA
- Norsk Data
- NS16000
- CRAY 1

La description complète du transport du système est donnée dans [Devin 85].



5 Les Pseudos-Instructions

- TITLE nom** [pseudo-instruction]
nom d'un module LLM3 dont c'est la première instruction.
- XREFI module,nom** [pseudo-instruction]
définit un nom de symbole externe, de nom **nom** qui doit se trouver dans une zone IMPURE du module **module**.
- XREFP module,nom** [pseudo-instruction]
définit un nom de symbole externe, de nom **nom** qui doit se trouver dans une zone PURE du module **module**.
- XDEFI nom** [pseudo-instruction]
le nom **nom**, qui doit se trouver dans une zone IMPURE du module courant devient externe et pourra être référencé dans d'autres modules au moyen de la pseudo-instruction XREFI.
- XDEFP nom** [pseudo-instruction]
le nom **nom**, qui doit se trouver dans une zone PURE du module courant devient externe et pourra être référencé dans d'autres modules au moyen de la pseudo-instruction XREFP.
- LABEL** [pseudo-instruction]
permet de définir une étiquette sans utiliser le champ **codop** d'une instruction.
- FENTRY nom,type** [pseudo-instruction]
nom est le nom d'un symbole global qui doit être attaché à une fonction LISP au moyen de l'instruction MAKFNT. **type** est un des types fonctionnels de LE_LISP qui peut être : SUBR0, SUBR1, SUBR2, SUBR3, SUBRN, SUBRF.
- ADR adr** [pseudo-instruction]
réserve 1 mot contenant la valeur **adr**.
- PURE** [pseudo-instruction]
Début des instructions. Tout ce qui suit ne sera jamais modifié et peut résider dans des ROM ou des zones mémoires protégées en écriture.
- IMPURE** [pseudo-instruction]
Début des données. Tout ce qui suit peut être modifié et doit résider dans une zone mémoire non protégée en écriture.
- END** [pseudo-instruction]
fin d'un module LLM3. C'est la dernière instruction de tout module.

6 Les Instructions de base

6.1 Les transferts de pointeurs

En plus de l'instruction MOV qui permet de transférer n'importe quel objet LISP, il existe d'autres instructions spécialisées dans le transfert de pointeurs ou d'octets contenus dans le tas (HEAP) ou dans la pile.

MOV op1,op2 *op1 -> op2* [instruction]

transfère le pointeur contenu dans l'opérande source **op1** dans l'opérande destination **op2**.

6.2 Les comparaisons de pointeurs

CABEQ op1,op2,lab *si op1=op2 vers lab* [instruction]

réalise une comparaison de pointeurs. Si l'opérande **op1** est égal à l'opérande **op2**, branchement à l'étiquette **lab**.

CABNE op1,op2,lab *si op1<>op2 vers lab* [instruction]

réalise une comparaison de pointeurs. Si l'opérande **op1** n'est pas égal à l'opérande **op2**, branchement à l'étiquette **lab**.

6.3 Le contrôle

BRA lab *lab -> PC* [instruction]

branchement à l'étiquette **lab** qui doit être à l'intérieur du module LLM3.

JMP lab *lab -> PC* [instruction]

branchement à l'étiquette **lab** qui peut être dans une autre module et en ce cas déclarée au moyen de la pseudo-instruction XREFP.

BRI op *op -> PC* [instruction]

réalise un branchement indirect à l'adresse contenue dans l'opérande **op**.

BRX adr,op *adr[op] -> PC* [instruction]

réalise un aiguillage à travers une table d'étiquettes rangée à l'adresse **adr**. **op** est l'index à utiliser compté à partir de 0. La table réside toujours dans le même module LLM3 en zone PURE.

SOBGEZ op,lab *op-1 -> op; si op >= 0 vers lab* [instruction]

décrémente l'opérande **op**. Si cet opérande est encore plus grand ou égal à 0, branchement à l'étiquette **lab**.

SOBGTZ op,lab *op-1 -> op; si op > 0 vers lab* [instruction]

décrémente l'opérande **op**. Si cet opérande est encore strictement plus grand que 0, branchement à l'étiquette **lab**.

7 La Pile

LLM3 dispose d'une pile unique pour le contrôle et les données. Elle est utilisée comme opérande implicite dans des instructions spécialisées. Ces instructions qui utilisent le registre pointeur de pile **SP** ne nécessitent pas de sens de construction privilégié de la pile ni de test de débordement implicite de celle-ci. La pile occupe en général 4k objets.

7.1 Organisation mémoire

La pile est constituée d'une zone mémoire dont les adresses sont rangées dans 3 variables globales qui doivent être chargées avant le lancement du système LE_LISP.

BSTACK [variable globale]
contient l'adresse du début de la zone allouée à la pile

ESTACK [variable globale]
contient l'adresse de la fin de la zone allouée à la pile

MSTACK [variable globale]
contient une adresse proche de la fin de la zone allouée à la pile. Elle permet de tester les débordements de la pile avant qu'ils n'arrivent. Typiquement MSTACK contient 64 objets avant ESTACK.

7.2 Gestion du pointeur de pile

3 instructions permettent de manipuler explicitement le pointeur de pile **SP**.

STACK op *SP -> op* [instruction]
transfère le contenu actuel du pointeur de pile dans l'opérande **op**.

SSTACK op *op -> SP* [instruction]
transfère l'opérande **op** dans le pointeur de pile.

CHKSTK op,lab *si SP > op vers lab* [instruction]
teste le débordement de pile. Si **SP** >= **op** branchement à l'étiquette **lab**.
op est en général MSTACK ou ESTACK.

7.3 Pile de contrôle

4 instructions permettent de manipuler des adresses de retour stockées dans la pile.

CALL lab *PUSH PC; lab -> PC* [instruction]
rajoute en sommet de pile la valeur courante du compteur ordinal et fait un branchement à l'étiquette **lab** qui doit se trouver dans le module LLM3 courant.

JCALL lab *PUSH PC; lab -> PC* [instruction]
est identique à l'instruction précédente mais le branchement peut avoir lieu dans un autre module LLM3. Dans ce cas l'étiquette doit être déclarée XREFP.

CALLI op *PUSH PC; op -> PC* [instruction]
rajoute en sommet de pile la valeur courante du compteur ordinal et fait un branchement à l'adresse contenue dans l'opérande **op**.

RETURN POP PC [instruction]

dépile une adresse qui devient le nouveau compteur ordinal.

7.4 Pile de données

TOPST op (SP) -> op [instruction]

le sommet de la pile est copié dans l'opérande **op**. Le pointeur de pile reste inchangé.

TOPSTR accu,op (accu) -> op [instruction]

le sommet de la pile, dont l'adresse est dans l'opérande **accu** (qui est toujours un accumulateur) est copié dans l'opérande **op**. L'adresse de la pile contenue dans l'accumulateur reste inchangé.

XTOPST op (SP) <-> op [instruction]

le sommet de la pile est échangé avec l'opérande **op**. Le pointeur de pile reste inchangé.

XTOPSTR accu,op (accu) <-> op [instruction]

le sommet de la pile, dont l'adresse est dans l'opérande **accu** (qui est toujours un accumulateur) est échangé avec l'opérande **op**. L'adresse de la pile contenue dans l'accumulateur reste inchangé.

PUSH op [instruction]

empile l'opérande **op**. Le pointeur de pile est mis à jour.

POP op [instruction]

dépile le sommet de pile dans l'opérande **op**. Le pointeur de pile est mis à jour.

POPR accu,op [instruction]

le sommet de la pile, dont l'adresse est dans l'opérande **accu** (qui est toujours un accumulateur) est dépilé dans l'opérande **op**. Le pointeur de pile contenu dans l'accumulateur est mis à jour.

ADJSTK op [instruction]

ajuste le pointeur de pile de telle sorte que les **op** derniers objets empilés soient enlevés de la pile.

Pour les 2 instructions suivantes l'indice du sommet de pile vaut 0, celui du sous-sommet 1, du sous-sous sommet 2

MOVXSP op1,op2 op1 -> stack[op2] [instruction]

transfère l'opérande **op1** dans le **op2**ième emplacement de la pile.

XSPMOV op1,op2 stack[op1] -> op2 [instruction]

transfère l'objet contenu dans le **op1**ième emplacement de la pile dans l'opérande **op2**.

8 Le garbage-collector (GC)

Le récupérateur de mémoire a besoin d'instructions spéciales permettant de marquer (au moyen d'un bit dans le pointeur ou d'une table de bits externe) tout objet LISP (de type NUMB, FLOAT, VECTOR, STRING, SYMBOL ou CONS). Pour toutes ces instructions, l'opérande est toujours de type accumulateur **accu**.

STMARK accu [instruction]

marque l'objet LISP d'adresse **accu**.

BTMARK accu,lab [instruction]

si l'objet LISP d'adresse **accu** est marqué branchement à l'étiquette **lab**.

BFMARK accu,lab [instruction]

si l'objet LISP d'adresse **accu** n'est pas marqué branchement à l'étiquette **lab**.

TCMARK accu,lab [instruction]

si l'objet LISP d'adresse **accu** est marqué, il est démarqué et branchement à l'étiquette **lab**.

BTLISP op,lab [instruction]

si l'objet d'adresse **op** est un objet LISP, branchement à l'étiquette **lab**.

CONVTOK op1,op2,op3 *op2-op1/2-10->op3* [instruction]

retourne dans **op3**, le nombre de K résultant de la différence des adresses **op2 - op1**. Cette instruction est utilisée pour calculer les tailles restantes des différentes zones dans les statistiques du GC.

9 Les cellules de liste (CONS)

9.1 Organisation mémoire

Les cellules de liste sont rangées dans une zone mémoire spéciale dont les adresses sont dans 2 variables globales qui doivent être chargées avant le lancement du système LE_LISP :

BCONS [variable globale]

qui contient l'adresse du début de la zone physique des cellules de liste (l'adresse de la première cellule)

CCONS [variable globale]

qui contient l'adresse du début de la zone des cellules de liste à initialiser par LE_LISP. Si le lanceur n'initialise aucune cellule de liste, CCONS est égal à BCONS.

Le gestionnaire de mémoire a également besoin d'une instruction qui permet d'avancer dans la zone des cellules de listes et de tester sa fin. Cette instruction spéciale permet d'implanter la zone des cellules de liste très soupagement y compris sous une forme multi-segments.

NXCONS op,lab [instruction]

op (qui contient un pointeur sur une cellule de liste) pointe maintenant sur la cellule de liste physiquement suivant. Si le pointeur obtenu est toujours de type cellule de liste, branchement à l'étiquette **lab**. Si le pointeur était à la fin de la zone des cellules de liste, il n'y a pas de branchement.

Il existe également un pointeur **FREEL** qui contient la tête de la liste libre des cellules de liste. Il est souhaitable que **FREEL** soit un registre pour pouvoir macrogénérer facilement les instructions de type **xCONS**. De plus une macro spéciale permet d'écrire dans ce pointeur.

SFREEL op op -> FREEL [macro LLM3]

transfère dans l'opérande **op** la tête de la liste libre des cellules de liste.

9.2 Test du type cellule de liste

C'est le plus fréquent test de type. Il faut donc particulièrement le soigner par exemple en mettant **BCONS** dans un registre.

BTCONS op,lab si (CONSP op) vers lab [instruction]

si l'opérande **op** est de type cellule de liste, branchement à l'adresse **lab**.

BFCONS op,lab si (ATOM op) vers lab [instruction]

si l'opérande **op** n'est pas de type cellule de liste, branchement à l'adresse **lab**.

9.3 Accès aux champs des cellules de liste

au moyen d'opérandes de la machine LLM3. Ces opérandes ne sont valides que pour des pointeurs sur des cellules de liste. Le code du système LISP écrit en LLM3 s'assure toujours que **accu** contient bien un pointeur de type cellule de liste avant d'utiliser ces opérandes.

CAR(accu) [opérande]

CDR(accu) [opérande]

9.4 Création (allocation) d'une cellule de liste

au moyen de 2 instructions. S'il n'est plus possible d'allouer de nouvelles cellules, appel de la routine **GCCONS** automatiquement par ces instructions (la routine **GCCONS** est toutefois écrite en LLM3) :

CONS op1,op2 (op1 . op2) -> op2 [instruction]

alloue une cellule de liste dont le **CAR** sera chargé avec l'opérande **op1** et le **CDR** avec l'opérande **op2**. Le résultat (c'est-à-dire l'adresse de la cellule de liste allouée) se retrouve dans **op2**.

XCONS op1,op2 (op2 . op1) -> op2 [instruction]

alloue une cellule de liste dont le **CDR** sera chargé avec l'opérande **op1** et le **CAR** avec l'opérande **op2**. Le résultat (c'est-à-dire l'adresse de la cellule de liste allouée) se retrouve dans **op2**.

NCONS op (op . NIL) -> op [macro LLM3]

alloue une cellule de liste dont le **CAR** sera chargé avec l'opérande **op** et le **CDR** avec l'opérande **NIL**. Le résultat (c'est-à-dire l'adresse de la cellule de liste allouée) se retrouve dans **op**.

9.5 Bit invisible

pour les amateurs de sensations fortes : un bit *invisible* positionnable dans toutes les cellules de liste, directement dans l'un des pointeurs ou au moyen d'une table de bits. Ce bit est très utile pour coder efficacement les types étendus. Dans toutes ces instructions, l'opérande qui contient l'adresse de la cellule de liste est toujours un accumulateur **accu**.

STINVSBL *accu* [instruction]

marque la cellule de liste d'adresse ***accu***.

CLINVSBL *accu* [instruction]

démarque la cellule de liste d'adresse ***accu***.

BTINVSBL *accu,lab* [instruction]

si la cellule de liste d'adresse ***accu*** est marquée branchement à l'étiquette ***lab***

BFINVSBL *accu,lab* [instruction]

si la cellule de liste d'adresse ***accu*** n'est pas marquée branchement à l'étiquette ***lab***

10 NIL

NIL est un opérande de type mémoire dont la valeur est égale à la variable globale BSYMB. Pour des raisons d'efficacité il est souhaitable de mettre NIL dans un accumulateur. LLM3 fournit 3 Macros-LLM3 pour faciliter son utilisation :

MOVNIL *op* *NIL -> op* [macro LLM3]

transfère la valeur de NIL dans l'opérande ***op***.

BTNIL *op,lab* *si op = NIL vers lab* [macro LLM3]

si l'opérande ***op*** est égal à NIL branchement à l'adresse ***lab***.

BFNIL *op,lab* *si op <> NIL vers lab* [macro LLM3]

si l'opérande ***op*** n'est pas égal à NIL branchement à l'adresse ***lab***.

11 Les symboles

11.1 Organisation mémoire

Les symboles sont rangés dans une zone mémoire spéciale dont les adresses sont dans 2 variables globales qui doivent être chargées avant le lancement du système LE_LISP :

BSYMB [variable globale]

qui contient l'adresse du début de la zone physique des symboles (l'adresse du premier symbole, NIL)

CSYMB [variable globale]

qui contient l'adresse du début de la zone des symboles à initialiser par LE_LISP. Si le lanceur n'initialise aucun symbole statiquement, CSYMB est égal à BSYMB.

Le gestionnaire de mémoire a également besoin d'une instruction qui permet d'avancer dans la zone des symboles et de tester sa fin. Cette instruction spéciale permet d'implanter la zone des symboles très soupagement y compris sous une forme multi-segments.

NXSYMB op,lab [instruction]

op (qui contient un pointeur sur un symbole) pointe maintenant sur le symbole physiquement suivant. Si le pointeur obtenu est toujours de type symbole, branchement à l'étiquette **lab**. Si le pointeur était à la fin de la zone des symboles, il n'y a pas de branchement.

11.2 Test de type symbole**BTSYMB op,lab** si (SYMBOLP op) vers lab [instruction]

si l'opérande **op** est un symbole, branchement à l'étiquette **lab**.

BFSYMB op,lab si (NOT (SYMBOLP op)) vers lab [instruction]

si l'opérande **op** n'est pas un symbole, branchement à l'étiquette **lab**.

11.3 Accès aux différents champs d'un symbole

Au moyen d'opérandes de la machine LLM3. Ces opérandes ne sont valides que pour des pointeurs sur des symboles. Le code du système LISP écrit en LLM3 s'assure toujours que **accu** contient bien un pointeur de type symbole avant d'utiliser ces opérandes.

CVAL(accu) Cell VALue [opérande]

contient n'importe quel objet LISP.

PLIST(accu) Properties LIST [opérande]

contient une liste LISP.

FVAL(accu) Function VALue [opérande]

contient un objet LISP.

OVAL(accu) Object VALue [opérande]

contient un objet LISP

ALINK(accu) Atom LINK [opérande]

contient un pointeur sur le symbole suivant.

PKGK(accu) PacKaGe Cell [opérande]

contient le symbole du package.

PNAME(accu) Print NAME [opérande]

contient une chaîne LISP.

11.4 Instructions spécialisées

Il existe deux autres champs dans chaque symbole, le champ **FTYPE** et le champ **PTYPE**, qui ne contiennent que de petites valeurs entières dans l'intervalle [0 256[. Il est possible de compacter ces champs dont les accès sont donnés par les 4 instructions suivantes.

GFTYPE op1,op2 FTYPE(op1) -> op2 [instruction]**SFTYPE op1,op2** op1 -> FTYPE(op2) [instruction]**GPTYPE op1,op2** PTYPE(op1) -> op2 [instruction]**SPTYPE op1,op2** op1 -> PTYPE(op2) [instruction]

11.5 Création statique

MAKFNT lab,plen,pname [instruction]
 permet de créer statiquement un nouveau symbole. La fonction associée se trouve à l'étiquette **lab** (cette étiquette doit être déclarée pseudo-instruction FENTRY), le nom du symbole a pour taille **plen** et comme caractères **pname**.

MAKCST lab,plen,pname [instruction]
 permet de créer statiquement une nouvelle constante. Le nom LLM3 de cette constante est **lab**, le nom du symbole a pour taille **plen** et comme caractères **pname**.

11.6 Les variables

LE_LISP doit protéger un certain nombre de symboles : les *constantes symboliques*, comme T, || ... Les symboles qui n'en sont pas sont appelés *variables*. Il existe 3 instructions qui vont gérer ce sous-ensemble des symboles. Ces instructions utilisent la variable globale BVAR qui est initialisée par LISP.

BVAR [variable globale]

SETBVAR op op -> BVAR [instruction]
 charge **op** comme adresse de début des variables.

BTVAR op,lab si (VARIABLEP op) vers lab [instruction]
 si l'opérande **op** est une variable, branchement à l'étiquette **lab**.

BFVAR op,lab si (NOT (VARIABLEP op)) vers lab [instruction]
 si l'opérande **op** n'est pas une variable branchement à l'étiquette **lab**.

12 Les nombres

LLM3 utilise deux types de nombres :

- les nombres entiers (sur 16 bits)
- les nombres flottants (sur 32/48 ou 64 bits, dépendant de l'implantation).

12.1 Les nombres entiers sur 16 bits

les opérandes immédiats de type entier se décrivent :
 #nnnn ou #\$hhhh

12.1.1 L'organisation mémoire

La représentation de ces entiers dépend de la machine cible : pour les machines à mots d'au moins 32 bits il est souhaitable de ranger directement ces valeurs dans les pointeurs eux-mêmes. Pour les systèmes dans lequel cela est impossible (les pointeurs font 16 ou 18 bits), LE_LISP gère une zone spéciale (peut être vide) qui contiendra des valeurs numériques entières. Cet espace est appelé espace NUMB. Les adresses de ses bornes doivent être chargées avant le lancement du système LE_LISP :

BNUMB [variable globale]
qui contient l'adresse du début de la zone physique des valeurs entières.

CSYMB [variable globale]
qui contient l'adresse du début de la zone des valeurs entières à initialiser par LE_LISP. Si le lanceur n'initialise aucun entier statiquement, CNUMB est égal à BNUMB.

Le gestionnaire de mémoire a également besoin d'une instruction qui permet d'avancer dans la zone des nombres entiers et de tester sa fin. Cette instruction spéciale permet d'implanter la zone des nombres entiers très soupagement y compris sous une forme multi-segments.

NXNUMB op,lab [instruction]
op (qui contient un pointeur sur un nombre entier) pointe maintenant sur le nombre entier physiquement suivant. Si le pointeur obtenu est toujours de type nombre entier, branchement à l'étiquette **lab**. Si le pointeur était à la fin de la zone des nombres entiers, il n'y a pas de branchement.

12.1.2 Les tests de type

BTFIX op,lab *si (FIXP op) vers lab* [instruction]
si l'opérande **op** est un nombre entier branchement à l'étiquette **lab**.

BFFIX op,lab *si (NOT (FIXP op)) vers lab* [instruction]
si l'opérande **op** n'est pas un nombre entier branchement à l'étiquette **lab**.

BTNUMB op,lab [instruction]
si l'opérande **op** est un nombre entier rangé dans l'espace NUMB, branchement à l'étiquette **lab**.

BFNUMB op,lab [instruction]
si l'opérande **op** n'est pas un nombre rangé dans l'espace NUMB, branchement à l'étiquette **lab**.

12.1.3 Les instructions de calcul

Pour toutes les instructions de calcul sur les entiers il n'y a pas de distinction entre les types des nombres entiers. De plus, si le dernier argument **lab** est fourni, il y a branchement à cette étiquette si un débordement se produit. Si l'étiquette n'est pas fournie le test de débordement n'est pas effectué.

INCR op[,lab] $op + 1 \rightarrow op$ [instruction]

DECR op[,lab] $op - 1 \rightarrow op$ [instruction]

PLUS op1,op2[,lab] $op1 + op2 \rightarrow op2$ [instruction]

DIFF op1,op2[,lab] $op2 - op1 \rightarrow op2$ [instruction]

NEGATE op $-op \rightarrow op$ [instruction]
réalise une négation arithmétique.

TIMES op1,op2[,lab] $op1 * op2 \rightarrow op2$ [instruction]

QUO op1,op2 *op2 / op1 -> op2* [instruction]
calcule le quotient de la division entière.

REM op1,op2 *op2 op1 -> op2* [instruction]
calcule le reste de la division entière.

12.1.4 Les comparaisons numériques entières

CNBEQ op1,op2,lab *si op1=op2 vers lab* [instruction]
réalise une comparaison numérique entière. Si l'opérande **op1** est égal à l'opérande **op2**, branchement à l'étiquette **lab**.

CNBNE op1,op2,lab *si op1<>op2 vers lab* [instruction]
réalise une comparaison numérique entière. Si l'opérande **op1** n'est pas égal à l'opérande **op2**, branchement à l'étiquette **lab**.

CNBLE op1,op2,lab *si op1<=op2 vers lab* [instruction]
réalise une comparaison numérique entière. Si l'opérande **op1** est plus petit ou égal à l'opérande **op2**, branchement à l'étiquette **lab**.

CNBLT op1,op2,lab *si op1<op2 vers lab* [instruction]
réalise une comparaison numérique entière. Si l'opérande **op1** est plus petit que l'opérande **op2**, branchement à l'étiquette **lab**.

CNBGE op1,op2,lab *si op1>=op2 vers lab* [instruction]
réalise une comparaison numérique entière. Si l'opérande **op1** est plus grand ou égal à l'opérande **op2**, branchement à l'étiquette **lab**.

CNBGT op1,op2,lab *si op1>op2 vers lab* [instruction]
réalise une comparaison numérique entière. Si l'opérande **op1** est plus grand que l'opérande **op2**, branchement à l'étiquette **lab**.

12.1.5 Les instructions logiques

Les opérandes de ces intructions doivent toujours être des valeurs entières sur 16 bits. Le résultat de ces opérations est toujours sur 16 bits.

LAND op1,op2 *op1 AND op2 -> op2* [instruction]
il s'agit là de l'instruction ET LOGIQUE.

LOR op1,op2 *op1 OR op2 -> op2* [instruction]
il s'agit là de l'instruction OU INCLUSIF LOGIQUE.

LXOR op1,op2 *op1 XOR op2 -> op2* [instruction]
il s'agit là de l'instruction OU EXCLUSIF LOGIQUE.

LSHIFT op1,op2 *op1 LSHIFT op2 -> op2* [instruction]
décale l'opérande **op2** de **op1** positions et range le résultat dans l'opérande **op2**. Si l'opérande **op1** est positif, le décalage s'effectue à gauche (multiplication), s'il est négatif, il s'effectue à droite (division).

12.2 Les nombres flottants

Ces instructions, qui ne sont pas obligatoires dans une version réduite du système, possèdent en général un opérande de type **lab** qui désigne l'étiquette où il faut se brancher en cas de débordement ou d'exception arithmétique. Les valeurs des nombres flottants sont rangées dans une zone mémoire spéciale dont les adresses sont dans 3 variables globales qui doivent être chargées avant le lancement du système LE_LISP.

BFLOAT [variable globale]
qui contient l'adresse du début de la zone physique des nombres flottants.

CFLOAT [variable globale]
qui contient l'adresse du début de la zone des nombres flottants à initialiser par LE_LISP. Si le lanceur n'initialise aucun nombre flottant statiquement, CFLOAT est égal à BFLOAT.

Le gestionnaire de mémoire a également besoin d'une instruction qui permet d'avancer dans la zone des nombres flottants et de tester sa fin. Cette instruction spéciale permet d'implanter la zone des nombres flottants très soupagement y compris sous une forme multi-segments.

NXFLOAT op,lab [instruction]
op (qui contient un pointeur sur un nombre flottant) pointe maintenant sur le nombre flottant physiquement suivant. Si le pointeur obtenu est toujours de type nombre flottant, branchement à l'étiquette **lab**. Si le pointeur était à la fin de la zone des nombres flottants, il n'y a pas de branchement.

12.2.1 Les tests de type nombre flottant

BTFLOAT op,lab *si (FLOATP op) vers lab* [instruction]
si l'opérande **op** est un nombre flottant branchement à l'étiquette **lab**.

BFFLOAT op,lab *si (NOT (FLOATP op)) vers lab* [instruction]
si l'opérande **op** n'est pas un nombre flottant branchement à l'étiquette **lab**.

12.2.2 Conversions

FIX op *(FIX op) -> op* [instruction]
convertit l'opérande **op** de type flottant en un entier sur 16 bits.

FLOAT op *(FLOAT op) -> op* [instruction]
convertit l'opérande **op** de type entier sur 16 bits en un flottant.

CVFTOA op1,strg,op2 [instruction]
convertit le nombre flottant contenu dans l'opérande **op1** en une suite de caractères qui sera rangée dans la chaîne LISP **strg**. Le nombre total de caractères rangés sera donné dans l'opérande **op2**.

CVATOF strg,n,op,lab [instruction]
strg contient une chaîne LISP de **n** caractères. CVATOF retourne dans **op** la valeur flottante correspondante. Si la conversion n'est pas possible, branchement à l'étiquette **lab**.

12.2.3 Les instructions de bases

FPLUS op1,op2,lab *op1+op2->op2* [instruction]

FDIFF op1,op2,lab *op2-op1->op2* [instruction]

FTIMES *op1,op2,lab* $op1 * op2 \rightarrow op2$ [instruction]

FQUO *op1,op2,lab* $op2 / op1 \rightarrow op2$ [instruction]

12.2.4 Les comparaisons

CFBEQ *op1,op2,lab* si $op1 = op2$ vers *lab* [instruction]

CFBNE *op1,op2,lab* si $op1 \neq op2$ vers *lab* [instruction]

CFBLT *op1,op2,lab* si $op1 < op2$ vers *lab* [instruction]

CFBLE *op1,op2,lab* si $op1 \leq op2$ vers *lab* [instruction]

CFBGT *op1,op2,lab* si $op1 > op2$ vers *lab* [instruction]

CFBGE *op1,op2,lab* si $op1 \geq op2$ vers *lab* [instruction]

12.2.5 Les fonctions circulaires et mathématiques

SIN *op1,op2* $\sin(op1) \rightarrow op2$ [instruction]

COS *op1,op2* $\cos(op1) \rightarrow op2$ [instruction]

ASIN *op1,op2* $\arcsin(op1) \rightarrow op2$ [instruction]

ACOS *op1,op2* $\arccos(op1) \rightarrow op2$ [instruction]

ATAN *op1,op2* $\arctangente(op1) \rightarrow op2$ [instruction]

EXP *op1,op2* $e^{op1} \rightarrow op2$ [instruction]

LOG *op1,op2* $\log(op1) \rightarrow op2$ [instruction]

LOG10 *op1,op2* $\log_{10}(op1) \rightarrow op2$ [instruction]

POWER *op1,op2,op3* $op1^{op2} \rightarrow op3$ [instruction]

SQRT *op1,op2* $\sqrt{op1} \rightarrow op2$ [instruction]

12.3 Les nombres à précision variable

L'implantation des nombres à précision variable oblige à introduire des instructions arithmétiques sur des quantités de 16 bits non signées qui retournent 2 valeurs : poids forts et poids faibles du calcul.

EPLUS *op1,op2,op3,op4,op5* $op1 + op2 + op3 \rightarrow op4|op5$ [instruction]

ETIMES *op1,op2,op3,op4,op5,op6* $op1 * op2 + op3 + op4 \rightarrow op5|op6$ [instruction]

EDIVIDE *op1,op2,op3,op4,op5* $op1|op2 \text{ quo } op3 \rightarrow op4, op1|op2 \text{ rem } op3 \rightarrow op5$ [instruction]

ECOMP *op1,op2,lab1,lab2,lab3* [instruction]

si $op1 < op2$ vers *lab1*, si $op1 = op2$ vers *lab2*, si $op1 > op2$ vers *lab3*. La comparaison a lieu non signée.

13 Les Vecteurs de Pointeurs LISP

Ce type permet l'utilisation de listes linéaires nécessitant deux fois moins de mémoire que les cellules de listes classiques.

13.1 Organisation mémoire

Les vecteurs de pointeurs sont rangés dans une zone mémoire spéciale dont les adresses sont dans 2 variables globales qui doivent être chargées avant le lancement du système LE_LISP. Chaque vecteur de pointeurs est en fait un pointeur vers l'espace tas (ou HEAP) où seront rangés effectivement les différents éléments du vecteur. Ce pointeur vers le tas n'est pas un objet LISP.

BVECT [variable globale]

qui contient l'adresse du début de la zone physique des vecteurs de pointeurs.

CVECT [variable globale]

qui contient l'adresse du début de la zone des vecteurs de pointeurs à initialiser par LE_LISP. Si le lanceur n'initialise statiquement aucun vecteur de pointeurs, CVECT est égal à BVECT.

Le gestionnaire de mémoire a également besoin d'une instruction qui permet d'avancer dans la zone des vecteurs de pointeurs et de tester sa fin. Cette instruction spéciale permet d'implanter la zone des pointeurs de vecteurs très souples y compris sous une forme multi-segments.

NXVECT op,lab [instruction]

op (qui contient un pointeur sur un vecteur de pointeurs) pointe maintenant sur le vecteur de pointeur physiquement suivant. Si le pointeur obtenu est toujours de type vecteur de pointeurs, branchement à l'étiquette **lab**. Si le pointeur était à la fin de la zone des vecteurs de pointeurs, il n'y a pas de branchement.

13.2 Test de type vecteur de pointeur

BTVECT op,lab [instruction]

si **op** est un vecteur de pointeurs, branchement à l'étiquette **lab**.

BFVECT op,lab [instruction]

si **op** n'est pas un vecteur de pointeurs, branchement à l'étiquette **lab**.

13.3 Allocation

Il n'existe pas d'instruction spécialisée mais un sous-programme LLM3. Ce sous-programme de nom **MAKEVECT** dans le module STRING demande dans **A1** la taille du vecteur de pointeurs que l'on veut créer, dans **A2** l'atome avec lequel doivent être initialisés les éléments du vecteur et retourne dans **A1** un pointeur sur le vecteur alloué.

13.4 Accès aux éléments d'un vecteur de pointeurs

l'accès aux éléments d'un vecteur de pointeurs est réalisé au moyen des instructions de type pointeur sur le HEAP.

14 Les Chaînes de Caractères

14.1 Organisation mémoire

Les chaînes de caractères sont rangées dans une zone mémoire spéciale dont les adresses sont dans 2 variables globales qui doivent être chargées avant le lancement du système LE_LISP. Chaque chaîne de caractères est en fait un nouveau pointeur vers l'espace tas (ou HEAP) où seront rangés effectivement les différents caractères de la chaîne. Ce pointeur vers le tas n'est pas un objet LISP.

BSTRG [variable globale]

qui contient l'adresse du début de la zone physique des chaînes de caractères.

CSTRG [variable globale]

qui contient l'adresse du début de la zone des chaînes de caractères à initialiser par LE_LISP. Si le lanceur n'initialise aucune chaîne de caractères, CSTRG est égal à BSTRG.

Le gestionnaire de mémoire a également besoin d'une instruction qui permet d'avancer dans la zone des chaînes de caractères et de tester sa fin. Cette instruction spéciale permet d'implanter la zone des chaînes de caractères très soupagement y compris sous une forme multi-segments.

NXSTRG op,lab [instruction]

op (qui contient un pointeur sur une chaîne de caractères) pointe maintenant sur la chaîne de caractères suivante. Si le pointeur obtenu est toujours de type chaîne de caractères, branchement à l'étiquette **lab**. Si le pointeur était à la fin de la zone des chaînes de caractères, il n'y a pas de branchement.

14.2 Test de type chaîne de caractères

BTSTRG op,lab [instruction]

si **op** est une chaîne de caractères, branchement à l'étiquette **lab**.

BFSTRG op,lab [instruction]

si **op** n'est pas une chaîne de caractères, branchement à l'étiquette **lab**.

14.3 Allocation

Il n'existe pas d'instruction spécialisée mais un sous-programme LLM3. Ce sous-programme de nom **MAKESTRG** dans le module STRING demande dans **A1** le nombre de caractères de la chaîne que l'on veut créer, dans **A2** le code ASCII initial de tous ces caractères et retourne dans **A1** un pointeur sur la chaîne LISP allouée.

14.4 Accès aux caractères

l'accès aux caractères d'une chaîne de caractères est réalisé au moyen des instructions de type octet sur le HEAP.

15 Zone du tas (HEAP)

Cette zone contient les valeurs des objets LISP de taille variable comme les chaînes de caractères, les vecteurs de pointeurs et dans certaines implantations les nombres flottants. Un objet dans le tas contient, outre la valeur d'un objet LISP, la taille de cet objet ainsi qu'un pointeur arrière permettant de connaître l'objet LISP dont il est la valeur. Les adresses de type tas, de même taille que les objets de type ADR, ne doivent être utilisées qu'avec des opérandes de type mémoire (à l'exclusion des accumulateurs) sous peine de corrompre le GC!

15.1 Organisation

Les adresses de la zone spéciale réservée au tas sont contenues dans 2 variables globales qui doivent être chargées avant le lancement du système LE LISP.

BHEAP [variable globale]

qui contient l'adresse du début de la zone physique du tas.

CHEAP [variable globale]

qui contient l'adresse du début de la zone du tas à initialiser par LE LISP. Si le lanceur n'initialise aucun objet dans le tas, **CHEAP** est égal à **BHEAP**.

15.2 Autres instructions

Les objets de type tas n'étant pas des objets LISP, il n'existe pas d'instructions de test de type. Les transferts d'adresses de tas sont réalisés au moyen de l'instruction **MOV** uniquement avec des opérandes de type mémoire et les comparaisons avec les instructions de type **CABxx** entre 2 objets de type **HEAP**. Toutefois on teste la fin de cette zone au moyen de l'instruction suivante.

CHBLT op1,op2,lab [instruction]

si le pointeur sur le tas **op1** est plus petit que le pointeur sur le tas **op2**, branchement à l'étiquette **lab**.

15.3 La récupération de l'espace

Le récupérateur (GC) gère cette espace d'une manière différente des autres espaces. En particulier il réalise une compaction linéaire de cette zone quand elle devient pleine.

NXHB size,heap $heap+B(size) \rightarrow heap$ [instruction]

heap contient une adresse de la zone tas d'un objet de type chaîne de caractères de taille **size**. Après exécution de l'instruction, **heap** pointe sur le mot physiquement suivant du tas. La taille est donnée en octets.

NXHF heap $heap+F \rightarrow heap$ [instruction]

heap contient l'adresse de la valeur d'un nombre flottant de la zone **HEAP**. Après exécution de l'instruction, **heap** pointe sur le mot physiquement suivant du **HEAP**. Cette instruction n'a de sens que pour les systèmes qui rangent les valeurs des nombres flottants dans le tas.

NXHP size,heap $heap+P(size) \rightarrow heap$ [instruction]

heap contient l'adresse d'une valeur de la zone **HEAP** de type vecteur de pointeurs de taille **size**. Après exécution de l'instruction, **heap** pointe sur le mot physiquement suivant du **HEAP**. La taille est donnée en pointeurs.

HBLT heap1,heap2,heap3 [*heap1..heap2*]->[*heap3*... [instruction]

permet de transporter une zone du tas débutant à l'adresse de tas **heap1** et se terminant à l'adresse de tas **heap2** exclus, dans une zone du tas réceptrice débutant à l'adresse de tas **heap3**. Il n'y a jamais de recouvrement de zone et les opérandes demeurent inchangés.

15.4 Accès indexé

15.4.1 de type octet

HBXMOV strg,index,op *strg[index]*->*op* [instruction]

transfère le *index*ème caractère de la chaîne LISP **strg** dans l'opérande *op*.

HBMOVX op,strg,index *op*->*strg[index]* [instruction]

transfère l'opérande *op* dans le *index*ème caractère de la chaîne LISP **strg**.

HBMOVN n1,strg1,n2,strg2,n3 [instruction]

transfère une suite de *n1* caractères située dans la chaîne LISP **strg1** à partir de la position *n2* dans la chaîne LISP destination **strg2** à partir de la position *n3*.

HBTEQ n1,strg1,n2,strg2,n3,lab [instruction]

compare une suite de *n1* caractères située dans la chaîne LISP **strg1** à partir de la position *n2* avec la chaîne LISP **strg2** à partir de la position *n3*. Branchement à l'étiquette **lab** si ces 2 sous-chaînes sont égales.

MOVBM size,asource,strg [instruction]

transfère d'une suite de *size* octets d'une zone d'adresse **asource** dans la chaîne LISP **strg**.

15.4.2 de type pointeurs

HPXMOV vect,n,op *vect[n]*->*op* [instruction]

transfère le *n*ème élément du vecteur LISP **vect** dans l'opérande *op*.

HPMOVX op,vect,n *op*->*vect[n]* [instruction]

transfère l'opérande *op* dans le *n*ème élément du vecteur LISP **vect**.

HPMOVN n1,vect1,n2,vect2,n3 [instruction]

transfère une suite de *n1* pointeurs situés dans le vecteur LISP **vect1** à partir de la position *n2* dans le vecteur destination **vect2** à partir de la position *n3*.

15.5 Accès aux champs cachés

Chaque objet dans le HEAP possède 2 champs accessibles au moyen d'instructions spécialisées : le champ OBJ (pointeur arrière) et SIZE (taille dans une unité dépendante du type de l'objet).

HGSIZE op1,op2 *SIZE(op1)*->*op2* [instruction]

HSSIZE op1,op2 *op1->SIZE(op2)* [instruction]
HGOBJ op1,op2 *OBJ(op1)->op2* [instruction]
HSOBJ op1,op2 *op1->OBJ(op2)* [instruction]

16 Les Entrées/Sorties

Les entrées/sorties sont réalisées au moyen d'instructions LLM3 ce qui apporte une totale indépendance vis-à-vis du système d'exploitation hôte. Généralement ces instructions LLM3 appellent des sous-programmes écrits en langage machine, en Pascal ou en C.

16.1 Les instructions sur le canal terminal

Pour son utilisation interactive, LE_LISP doit pouvoir gérer complètement un terminal classique ou *bit-mapped*. Dans un système minimum qui ne contient pas d'entrées/sorties sur disque, seules les instructions TTYIN, TTYSTRG, TTYMSG et TTYCRLF sont nécessaires.

TTYIN op [instruction]

retourne le caractère suivant lu sur le terminal dans l'opérande **op**.
Attention : cette routine doit absolument retourner le code lu TEL QUEL (en particulier le RETURN ne doit pas être traduit en Line Feed, ni le [en M majuscule ...]). De plus le système hôte ne doit pas réaliser d'écho.

TTYIS op,cc [instruction]

Si un caractère est prêt à être lu sur le terminal il est retourné dans l'opérande **op** et le code d'erreur, retourné dans l'opérande **cc**, vaut 0. Si un caractère n'est pas prêt à être lu le code d'erreur **cc** vaut -1.

TTYMSG n,<suite-de-caractères> [instruction]

sort sur le terminal une suite de caractères dont la taille se trouve dans l'opérande **n** et les caractères dans le deuxième opérande encadrés du séparateur " .

TTYCRLF [instruction]

sort sur le terminal un saut de ligne.

TTYSTRG n,strg [instruction]

sort sur le terminal les **n** premiers caractères de la chaîne LISP **strg**.

16.2 Les instructions sur les fichiers

Toutes ces instructions utilisent un numéro de canal **chan** qui est alloué par le système LE_LISP et retournent un code d'erreur **cc** qui vaut 0 si l'instruction s'est correctement déroulée et un nombre dépendant du système hôte en cas d'erreur.

MAXCHAN [variable globale]

est une variable globale qui contient le nombre maximum de canaux autorisés par le système hôte. Cette variable doit être chargée avant le lancement du système LE_LISP.

INFILE chan,strg,cc [instruction]

ouvre en entrée sur le canal de numéro **chan** le fichier dont le nom est la chaîne LISP **strg**. Le code d'erreur **cc** vaut 0 si tout s'est bien passé (c'est à dire si le fichier existait).

OUFILE chan,strg,cc [instruction]

ouvre en sortie sur le canal de numéro **chan** le fichier dont le nom est la chaîne LISP **strg**. Si un fichier de même nom existait, son contenu est détruit. Le code d'erreur **cc** vaut 0 si tout s'est bien passé c'est-à-dire si le fichier n'est pas protégé en écriture.

APFILE chan,strg,cc [instruction]

ouvre en sortie sur le canal de numéro **chan** le fichier dont le nom est la chaîne LISP **strg**. Si un fichier de même nom existait déjà les sorties suivantes sur ce fichier seront ajoutées à la fin de celui-ci. Le code d'erreur **cc** vaut 0 si tout s'est bien passé c'est-à-dire si le fichier n'est pas protégé en écriture.

INBF chan,strg,siz,cc [instruction]

lit sur le canal **chan** la ligne suivante dont les caractères seront rangés dans la chaîne LISP **strg**. La ligne se termine par une marque fin de ligne, dépendante du système, qui ne doit pas être transférée. Au retour **size** contient le nombre de caractères transférés dans la chaîne LISP. Le code d'erreur **cc** vaut 0 si tout s'est bien passé, 1 si la fin du fichier a été détectée et 2 si la taille de la ligne lue dépasse la taille de la chaîne.

OUTF chan,strg,size,cc [instruction]

écrit sur le canal de numéro **chan** la chaîne LISP **strg** sur **size** caractères puis écrit une marque de fin de ligne. Cette marque dépend du système d'exploitation (LF pour UNIX, CR/LF pour VMS ...). Au retour le code d'erreur **cc** vaut 0 si tout s'est bien passé.

OUTFL chan,strg,size,cc [instruction]

écrit sur le canal de numéro **chan** la chaîne LISP **strg** sur **size** caractères. Aucune autre information supplémentaire n'est écrite. Au retour le code d'erreur **cc** vaut 0 si tout s'est bien passé.

FCLOS chan,cc [instruction]

ferme le canal de numéro **chan**. Au retour le code d'erreur **cc** vaut 0 si tout s'est bien passé.

FDELE strg,cc [instruction]

détruit un fichier dont le nom est la chaîne LISP **strg**. Au retour le code d'erreur **cc** vaut 0 si tout s'est bien passé.

FRENA strg1,strg2,cc [instruction]

renomme un fichier dont le nom est la chaîne LISP **strg1** dans le nom **strg2**. Au retour le code d'erreur **cc** vaut 0 si tout s'est bien passé.

16.3 Les instructions sur les images mémoire

Ces deux instructions permettent de manipuler des images-mémoire. Une image mémoire contient l'ensemble des zones dynamiques du système.

CORSAV strg,cc [instruction]

sauve une image mémoire dans un fichier dont le nom est la chaîne LISP **strg**. Au retour le code d'erreur **cc** vaut 0 si tout s'est bien passé.

COREST strg,cc [instruction]

restitue une image mémoire à partir d'un fichier dont le nom est la chaîne LISP **strg**. Au retour le code d'erreur **cc** vaut 0 si tout s'est bien passé.

17. Les instructions système

Ces instructions sont dépendantes du système d'exploitation utilisé et ne sont pas obligatoires pour faire fonctionner le système LE_LISP.

CLINE strg [instruction]

envoie au système d'exploitation hôte la commande qui se trouve dans la chaîne LISP **strg**.

RUNTIME op [instruction]

retourne dans l'opérande **op** le temps CPU utilisé depuis le lancement du système LE_LISP. Ce temps doit être donné en secondes. En fonction du système, RUNTIME peut retourner soit un nombre entier, soit un nombre flottant pour avoir une bonne précision.

SLEEP op [instruction]

endort le système **op** secondes. **op** est donné en secondes flottantes.

INTON [instruction]

active les interruptions système durant l'exécution de LISP.

INTOFF [instruction]

inhibe les interruptions système durant l'exécution de LISP.

INTEST [instruction]

teste si une interruption est présente. Cette instruction est utilisée dans les matériels qui ne possèdent pas d'interruptions matérielles.

GETENVRN strg1,strg2,op [instruction]

demande au système la valeur de la *variable système* de nom **strg1**. S'il le peut, le système renvoie une réponse dans la chaîne de caractères **strg2** sur **op** caractères.

GETGLOBAL strg,op [instruction]

range dans **op** l'adresse d'un symbole global dont le nom est dans la chaîne de caractères **<strg>**.

CALLG op1,op2 [instruction]

appelle un sous-programme externe dont **op1** arguments sont empilés et retourne la valeur dans **op2**.

18 Les instructions d'accès à la mémoire

Ces instructions ne sont utilisées que par le chargeur mémoire associé au compilateur. Elles ne sont pas nécessaires pour un système uniquement interprété.

ADRHL op1,op2,op3 [instruction]

op1 contient un objet LISP. ADRHL charge dans **op2** les poids forts et dans **op3** les poids faibles de l'adresse de cet objet sous la forme de petits entiers LISP.

HLADR op1,op2,op3 [instruction]

op1 et **op2** contiennent respectivement les poids forts et les poids faibles de l'adresse d'un objet LISP. HLADR reconstruit cette adresse et la range dans **op3**.

MEMSET op1,op2 *op1 -> [op2]* [instruction]

transfère la valeur **op1** dans la mémoire d'adresse **op2**. La taille de la mémoire dépend de la machine cible, c'est typiquement la plus petite partie de la mémoire aisément accessible. Cette instruction correspond exactement à la fonction LISP MEMORY qui est utilisée dans le chargeur du compilateur. *Par exemple sur VAX on transfère un octet, sur 68000 on transfère un mot de 16 bits ...*

MEMGET op1,op2 [*op1*] -> *op2* [instruction]

transfère la valeur de la mémoire d'adresse **op1** dans l'opérande **op2**. Comme pour MEMSET la longueur de la valeur transférée est dépendante de la machine.

19 Les Fonctions

19.1 Les types des fonctions

LE_LISP est un ensemble de fonctions de 2 types :

- les SUBR qui évaluent leurs arguments
- les FSUBR qui ne les évaluent pas

19.2 Règle d'appel des fonctions

pour les SUBR à 0,1,2 ou 3 arguments (SUBR0, SUBR1, SUBR2, SUBR3), les arguments sont passés respectivement dans les accumulateurs : A1, A2, A3. Pour les SUBR à plus d'arguments ou à nombre variable (SUBRN), les arguments sont tous empilés et le nombre d'arguments total est fourni dans A4. Pour les SUBRF, la liste des arguments non évalués (i.e. le CDR de l'appel) se trouve dans A1. Toutes les fonctions retournent une valeur dans A1.

20 Programmer en LLM3

20.1 Exemple

Ce premier exemple montre la traduction de la fonction LISP REMQ, dans un style récursif. Voici la définition de cette fonction en LISP :

```
(DE REMQ (a l)
  (COND ((ATOM l) ())
        ((EQ a (CAR l)) (REMQ a (CDR l)))
        (T (CONS (CAR l) (REMQ a (CDR l))))))
```

et en LLM3

```
;      Début du module

      TITLE      module

      .....      ; déclarations

      MAKFNT      REMQ.#4,"remq"      ; définition du symbole
      .....      ; autres déclarations

      FENTRY      REMQ.SUBR2      ; déclaration du point d'entrée

      BTCONS      A2,REMQ1      ; la liste est vide
      MOVNIL      A1      ; s'assure d'une valeur NIL
      RETURN      ; et rentre.
REMQ1      MOV      CAR(A2),A3      ; A3 <- élément suivant de l
      MOV      CDR(A2),A2      ; A2 <- le reste de la liste
      CABEQ      A3,A1,REMQ      ; c'est un élément à ne pas copier.
      PUSH      A3      ; sauve l'élément à construire
      CALL      REMQ      ; récurse sur le reste de la liste!
      POP      A3
      CONS      A3,A1      ; construction récursive en retour
      RETURN

      .....

      END      ; fin du module
```

20.2 traitement des SUBRN

Voici le modèle du traitement des SUBRN dans l'ordre des arguments :

```
FENTRY      FOO.SUBRN
;----
      MOV      A4,A3
      BRA      TESTFIN
NEXTARG      XSPMOV      A4,A1
;      ....      traitement de A1
TESTFIN      SOBGEZ      A4,NEXTARG
      ADJSTK      A3
      RETURN
```

et dans l'ordre inverse des arguments :

```
FENTRY      BAR.SUBRN
;----
      BRA      TESTFIN
NEXTARG      POP      A1
```

```

      ....      traitement de A1
TESTFIN  SOBGEZ  A4,NEXTARG
         RETURN

ex : la fonction PLUS

      FENTRY  PLUS,SUBRN
:---
      MOV      #0,A1      ; (+) -> 0
      BRA      PLUS2
PLUS1    POP      A2      ; argument suivant
      BFFIX    A2,PLUSERR1 ; il faut un nb!
      PLUS     A2,A1,PLUSOVF
PLUS2    SOBGEZ  A4,PLUS1
         RETURN
PLUSERR1 MOV      A2,A1
         MOV     .PLUS,A2
         JMP     ERRN1A
PLUSOVF  MOV      .PLUS,A2
         JMP     ERROVF

```

20.3 L'oblist (la liste des symboles)

est gérée au moyen d'un tableau de hachage représenté par un vecteur de pointeurs. Les collisions sont gérées par chaînage au moyen du champ ALINK des symboles. La fin d'un tel chaînage est indiquée par une valeur numérique qui correspond au numéro de l'entrée dans la table.

HASHTAB [variable]

contient l'adresse du vecteur de pointeurs de la table de hachage.

Voici un exemple de parcours de l'OBLIST

ex : la fonction OBLIST la plus simple
(sans le traitement des packages) :

```

      XREFI      READ,HASHTAB
      ...
      FENTRY    SIMPOBLIST,SUBRO
:---
OBLIST  MOVNIL   A1      ; liste résultat
      HGSIZE    HASHTAB,A4 ; taille de la table
      DECR      A4
OBLIST1 HPXMOV   HASHTAB,A4,A4 ; A4 symbole suivant
      BRA      OBLIST8      ; vers le test
OBLIST2 MOV      A4,A3      ; pour le test d'arrêt 3D
      BTSTRG    PNAME(A4),OBLIST5 ; pas de lien 3D
OBLIST3 MOV      PNAME(A4),A4 ; avance 3D
OBLIST5 CONS     A4,A1      ; rajoute en tete
OBLIST7 CABNE    A4,A3,OBLIST3 ; boucle en 3D
      MOV      ALINK(A4),A4 ; avance en 2D
OBLIST8 BTSYMB   A4,OBLIST2 ; il y en a encore
      SOBGEZ    A4,OBLIST1 ; bucket suivant
      RETURN

```

21 Occurrences Statiques du noyau de l'interprete

Occurrences statiques des pseudos-instructions

Nombre de pseudos-instructions : 2352

1	ADR	207	8.8 %
2	END	14	0.5 %
3	ENDC	58	2.4 %
4	FENTRY	399	16.9 %
5	IFEQ	10	0.4 %
6	IFNE	48	2.0 %
7	IMPURE	13	0.5 %
8	LABEL	414	17.6 %
9	PURE	21	0.8 %
10	TITLE	14	0.5 %
11	XDEFI	123	5.2 %
12	XDEFP	514	21.8 %
13	XREFI	213	9.0 %
14	XREFP	304	12.9 %

Occurrences statiques des instructions (alpha)

Nombre d'instructions : 7925

1	ACOS	1	0.0 %
2	ADJSTK	17	0.2 %
3	ADRHL	1	0.0 %
4	APFILE	1	0.0 %
5	ASIN	1	0.0 %
6	ATAN	1	0.0 %
7	BFCNS	122	1.5 %
8	BFFIX	180	2.2 %
9	BFFLOAT	5	0.0 %
10	BFINVSBL	5	0.0 %
11	BFMARK	4	0.0 %
12	BFNIL	51	0.6 %
13	BFNUMB	3	0.0 %
14	BFSTRG	8	0.1 %
15	BFSYMB	35	0.4 %
16	BFVAR	17	0.2 %
17	BFVECT	11	0.1 %
18	BRA	470	5.9 %
19	BRI	30	0.3 %
20	BRX	9	0.1 %
21	BTCONS	69	0.8 %
22	BTFIX	23	0.2 %
23	BTFLOAT	66	0.8 %
24	BTINVSBL	2	0.0 %
25	BTLISP	1	0.0 %

26	BTMARK	1	0.0 %
27	BTNIL	97	1.2 %
28	BTSTRG	25	0.3 %
29	BTSYMB	13	0.1 %
30	BTVAR	7	0.0 %
31	BTVECT	7	0.0 %
32	CABEQ	45	0.5 %
33	CABNE	41	0.5 %
34	CALL	339	4.2 %
35	CALLG	1	0.0 %
36	CFBEQ	1	0.0 %
37	CFBGE	2	0.0 %
38	CFBGT	1	0.0 %
39	CFBLE	1	0.0 %
40	CFBLT	1	0.0 %
41	CFBNE	1	0.0 %
42	CHBLT	5	0.0 %
43	CHKSTK	18	0.2 %
44	CLINE	1	0.0 %
45	CLINVSBL	3	0.0 %
46	CNBEQ	180	2.0 %
47	CNBGE	39	0.4 %
48	CNBGT	26	0.3 %
49	CNBLE	27	0.3 %
50	CNBLT	64	0.8 %
51	CNBNE	83	1.0 %
52	CONS	84	1.0 %
53	CONVTOK	2	0.0 %
54	COREST	1	0.0 %
55	CORSAV	1	0.0 %
56	COS	1	0.0 %
57	CVATOF	1	0.0 %
58	CVFTOA	2	0.0 %
59	DECR	47	0.5 %
60	DIFF	35	0.4 %
61	ECOMP	3	0.0 %
62	EDIVIDE	3	0.0 %
63	EPLUS	8	0.1 %
64	ETIMES	3	0.0 %
65	EXP	1	0.0 %
66	FCLOS	3	0.0 %
67	FDELE	1	0.0 %
68	FDIFF	4	0.0 %
69	FIX	2	0.0 %
70	FLOAT	55	0.6 %
71	FPLUS	5	0.0 %
72	FQUO	3	0.0 %
73	FRENA	1	0.0 %
74	FTIMES	2	0.0 %
75	GETENVRN	1	0.0 %
76	GETGLOBAL	1	0.0 %
77	GFTYPE	10	0.1 %
78	GPTYPE	2	0.0 %
79	HBLT	3	0.0 %
80	HEMOVN	10	0.1 %
81	HEMOVX	56	0.7 %
82	HBTEQ	4	0.0 %
83	HBXMOV	43	0.5 %
84	HGOBJ	1	0.0 %
85	HGSIIZE	59	0.7 %
86	HLADR	2	0.0 %

87	HPMOV	1	0.0 %
88	HPMOVX	84	1.0 %
89	HPXMOV	50	0.6 %
90	HSOBJ	4	0.0 %
91	HSSIZE	4	0.0 %
92	INBF	1	0.0 %
93	INCR	91	1.1 %
94	INFILE	1	0.0 %
95	INTEST	29	0.3 %
96	INTOFF	2	0.0 %
97	INTON	4	0.0 %
98	JCALL	109	1.3 %
99	JMP	233	2.9 %
100	LAND	8	0.1 %
101	LOG	1	0.0 %
102	LOG10	1	0.0 %
103	LOR	3	0.0 %
104	LSHIFT	9	0.1 %
105	LXOR	3	0.0 %
106	MAKCST	113	1.4 %
107	MAKFNT	407	5.1 %
108	MEMGET	1	0.0 %
109	MEMSET	1	0.0 %
110	MOV	1862	23.4 %
111	MOVBM	1	0.0 %
112	MOVNIL	174	2.1 %
113	MOVXSP	8	0.1 %
114	NCONS	53	0.6 %
115	NEGATE	13	0.1 %
116	NOP	4	0.0 %
117	NXCONS	2	0.0 %
118	NXFLOAT	2	0.0 %
119	NXHB	5	0.0 %
120	NXHF	2	0.0 %
121	NXHP	5	0.0 %
122	NXNUMB	2	0.0 %
123	NXSTRG	2	0.0 %
124	NXSYMB	1	0.0 %
125	NXVECT	2	0.0 %
126	OUFILE	1	0.0 %
127	OUTF	1	0.0 %
128	OUTFL	1	0.0 %
129	PLUS	52	0.6 %
130	POP	521	6.5 %
131	POPR	30	0.3 %
132	POWER	1	0.0 %
133	PUSH	552	6.9 %
134	QUO	5	0.0 %
135	REM	9	0.1 %
136	RETURN	492	6.2 %
137	RUNTIME	3	0.0 %
138	SETHVAR	1	0.0 %
139	SFREEL	2	0.0 %
140	SFTYPE	9	0.1 %
141	SIN	1	0.0 %
142	SLEEP	1	0.0 %
143	SOBGEZ	57	0.7 %
144	SOBGTZ	23	0.2 %
145	SPTYPE	5	0.0 %
146	SQRT	1	0.0 %
147	SSTACK	11	0.1 %

148	STACK	22	0.2 %
149	STINVSBL	4	0.0 %
150	STMARK	9	0.1 %
151	TCMARK	8	0.1 %
152	TIMES	9	0.1 %
153	TOPST	23	0.2 %
154	TOPSTR	3	0.0 %
155	TTYCRLF	15	0.1 %
156	TTYIN	3	0.0 %
157	TTYIS	1	0.0 %
158	TTYMSG	61	0.7 %
159	TTYSTRG	15	0.1 %
160	XCONS	13	0.1 %
161	XSPMOV	22	0.2 %
162	XTOPST	35	0.4 %
163	XTOPSTR	3	0.0 %

Occurrences statiques des instructions (par nombre)

Nombre d'instructions : 7925

1	MOV	1862	23.4 %
2	PUSH	552	6.9 %
3	POP	521	6.5 %
4	RETURN	492	6.2 %
5	BRA	470	5.9 %
6	MAKFNT	407	5.1 %
7	CALL	339	4.2 %
8	JMP	233	2.9 %
9	BFFIX	180	2.2 %
10	MOVNIL	174	2.1 %
11	CNBEQ	160	2.0 %
12	BFCONS	122	1.5 %
13	MAKCST	113	1.4 %
14	JCALL	109	1.3 %
15	BTNIL	97	1.2 %
16	INCR	91	1.1 %
17	CONS	84	1.0 %
18	HPMOVX	84	1.0 %
19	CNBNE	83	1.0 %
20	BTCONS	69	0.8 %
21	BTFLOAT	66	0.8 %
22	CNBLT	64	0.8 %
23	TTYMSG	61	0.7 %
24	HGSIZE	59	0.7 %
25	SOBGEZ	57	0.7 %
26	HEMOVX	56	0.7 %
27	FLOAT	55	0.6 %
28	NCONS	53	0.6 %
29	PLUS	52	0.6 %
30	BFNIL	51	0.6 %
31	HPXMOV	50	0.6 %
32	DECR	47	0.5 %
33	CABEQ	45	0.5 %
34	HEXMOV	43	0.5 %
35	CABNE	41	0.5 %
36	CNBGE	39	0.4 %

37	BFSYMB	35	0.4 %
38	DIFF	35	0.4 %
39	XTOPST	35	0.4 %
40	BRI	30	0.3 %
41	POPR	30	0.3 %
42	INTEST	29	0.3 %
43	CNBLE	27	0.3 %
44	CNBGT	26	0.3 %
45	BTSTRG	25	0.3 %
46	BTFIX	23	0.2 %
47	SOBGTZ	23	0.2 %
48	TOPST	23	0.2 %
49	STACK	22	0.2 %
50	XSPMOV	22	0.2 %
51	CHKSTK	18	0.2 %
52	ADJSTK	17	0.2 %
53	BFVAR	17	0.2 %
54	TTYCRLF	15	0.1 %
55	TTYSTRG	15	0.1 %
56	BTSYMB	13	0.1 %
57	NEGATE	13	0.1 %
58	XCONS	13	0.1 %
59	BFVECT	11	0.1 %
60	SSTACK	11	0.1 %
61	GFTYPE	10	0.1 %
62	HBMVM	10	0.1 %
63	BRX	9	0.1 %
64	LSHIFT	9	0.1 %
65	REM	9	0.1 %
66	SFTYPE	9	0.1 %
67	STMARK	9	0.1 %
68	TIMES	9	0.1 %
69	BFSTRG	8	0.1 %
70	EPLUS	8	0.1 %
71	LAND	8	0.1 %
72	MOVXSP	8	0.1 %
73	TCMARK	8	0.1 %
74	BTVAR	7	0.0 %
75	BTVECT	7	0.0 %
76	BFFLOAT	5	0.0 %
77	BFINVSBL	5	0.0 %
78	CHBLT	5	0.0 %
79	FPLUS	5	0.0 %
80	NXHB	5	0.0 %
81	NXHP	5	0.0 %
82	QUO	5	0.0 %
83	SPTYPE	5	0.0 %
84	BFMARK	4	0.0 %
85	FDIFF	4	0.0 %
86	HBTEQ	4	0.0 %
87	HSOBJ	4	0.0 %
88	HSSIZE	4	0.0 %
89	INTON	4	0.0 %
90	NOP	4	0.0 %
91	STINVSBL	4	0.0 %
92	BFNUMB	3	0.0 %
93	CLINVSBL	3	0.0 %
94	ECOMP	3	0.0 %
95	EDIVIDE	3	0.0 %
96	ETIMES	3	0.0 %
97	FCLOS	3	0.0 %

98	FQUO	3	0.0 %
99	HBLT	3	0.0 %
100	LOR	3	0.0 %
101	LXOR	3	0.0 %
102	RUNTIME	3	0.0 %
103	TOPSTR	3	0.0 %
104	TTYIN	3	0.0 %
105	XTOPSTR	3	0.0 %
106	BTINVSBL	2	0.0 %
107	CFBGE	2	0.0 %
108	CONVTOK	2	0.0 %
109	CVFTOA	2	0.0 %
110	FIX	2	0.0 %
111	FTIMES	2	0.0 %
112	GPTYPE	2	0.0 %
113	HLADR	2	0.0 %
114	INTOFF	2	0.0 %
115	NXCONS	2	0.0 %
116	NXFLOAT	2	0.0 %
117	NXHF	2	0.0 %
118	NXNUMB	2	0.0 %
119	NXSTRG	2	0.0 %
120	NXVECT	2	0.0 %
121	SFREEEL	2	0.0 %
122	ACOS	1	0.0 %
123	ADRHL	1	0.0 %
124	APFILE	1	0.0 %
125	ASIN	1	0.0 %
126	ATAN	1	0.0 %
127	BTLISP	1	0.0 %
128	BTMARK	1	0.0 %
129	CALLG	1	0.0 %
130	CFBEQ	1	0.0 %
131	CFBGT	1	0.0 %
132	CFBLE	1	0.0 %
133	CFBLT	1	0.0 %
134	CFBNE	1	0.0 %
135	CLINE	1	0.0 %
136	COREST	1	0.0 %
137	CORSAV	1	0.0 %
138	COS	1	0.0 %
139	CVATOF	1	0.0 %
140	EXP	1	0.0 %
141	FDELE	1	0.0 %
142	FRENA	1	0.0 %
143	GETENVRN	1	0.0 %
144	GETGLOBAL	1	0.0 %
145	HGOBJ	1	0.0 %
146	HPMOVN	1	0.0 %
147	INBF	1	0.0 %
148	INFILE	1	0.0 %
149	LOG	1	0.0 %
150	LOG10	1	0.0 %
151	MEMGET	1	0.0 %
152	MEMSET	1	0.0 %
153	MOVBN	1	0.0 %
154	NXSYMB	1	0.0 %
155	OUFILE	1	0.0 %
156	OUTF	1	0.0 %
157	OUTFL	1	0.0 %
158	POWER	1	0.0 %

159	SETBVAR	1	0.0 %
160	SIN	1	0.0 %
161	SLEEP	1	0.0 %
162	SQRT	1	0.0 %
163	TTYIS	1	0.0 %

Occurrences statiques des opérandes

Nombre d'opérandes : 13383

1	A1	2349	17.5 %
2	A2	1622	12.1 %
3	A3	937	7.0 %
4	A4	987	7.3 %
5	ALINK	38	0.2 %
6	CAR	169	1.2 %
7	CDR	242	1.8 %
8	CONST	61	0.4 %
9	CVAL	115	0.8 %
10	FVAL	31	0.2 %
11	IMAD	99	0.7 %
12	IMAT	558	4.1 %
13	LABEL	4150	31.0 %
14	MEMADR	940	7.0 %
15	NUMB	958	7.1 %
16	PKGC	40	0.2 %
17	PLIST	11	0.0 %
18	PNAME	47	0.3 %
19	VAL	23	0.1 %

22 Occurrences Dynamiques

Cette section contient des statistiques dynamiques de l'utilisation de la machine LLM3.

Chaque série de statistiques contient quatre tableaux. Les éléments vides du tableau ne sont pas imprimés.

Le premier tableau contient le nombre d'instructions réellement exécutées, triées par ordre alphabétique. Le nombre d'instructions est donné en K instructions avec une précision de 1K.

Le second tableau contient le nombre d'instructions réellement exécutées, triées par fréquence. Le nombre d'instructions est donné en K instructions avec une précision de 1K.

Le troisième tableau contient les fréquences des opérandes.

- A1/A2/A3/A4 : les 4 accumulateurs de LLM3.
- VAL : accès indirect
- CAR/CDR : accès aux constituants des cellules de liste.
- CVAL/PLIST/FVAL/ALINK/OVAL/PKGC/PNAME : accès aux constituants des symboles.
- IMAT : accès à l'adresse d'un symbole

- IMAD : accès à l'adresse d'une étiquette
- MEMADR : accès à un mot mémoire déclaré ADR
- NUMB : valeur immédiate
- PCMOD : nombre de fois où le compteur ordinal (PC) a été modifié. Ce nombre est à rapprocher non pas du nombre d'opérandes mais du nombre d'instructions exécutées.

Le quatrième tableau est un récapitulatif du nombre dynamique d'accès mémoire. Il contient les champs suivants :

- CODE : nombre d'instructions LLM3 lues; on suppose qu'une instruction LLM3 tient sur un mot.
- STACK : nombre d'accès à la pile (contrôle et donnée confondus)
- MEMADR : nombre d'accès à des mots de travail du système (déclarés en LLM3 au moyen de la pseudo ADR)
- OBJ : nombre d'accès à la mémoire des objets LISP (symboles, nombres, objet LISP chaîne, objet LISP vecteur)
- OBJIND : nombre d'accès aux valeurs indirectes du tas.

22.1 Le chargement de l'environnement initial

Ces premiers résultats sont obtenus après avoir chargé les fichiers :

- ../llib/startup.ll
- ../llib/macroch.ll
- ../llib/defs.ll
- ../llib/virtty.ll
- ../lltool/llm3.ll
- ../lltool/statdyn.ll

Il s'agit donc d'un test de lecture de fichiers LISP. Ce qui représente à peu près 2000 lignes de LISP, ou 70k octets de programme source. 8000 CONS ont été alloués, 500 symboles et 600 chaînes de caractères ont été créés.

Occurrences dynamiques des instructions (alpha)

Nombre d'instructions (en K instructions) : 5072

1	ADJSTK	15	0.2 %
2	BFCNS	42	0.8 %
3	BFFIX	13	0.2 %
4	BFNIL	228	4.4 %
5	BFSYMB	67	1.3 %
6	BRA	92	1.8 %
7	BRI	20	0.3 %
8	BRX	103	2.0 %
9	BTCONS	111	2.1 %
10	BTFIX	4	0.0 %
11	BTNIL	137	2.7 %
12	BTSTRG	21	0.4 %
13	BTSYMB	3	0.0 %
14	BTVAR	16	0.3 %
15	CABEQ	51	1.0 %
16	CABNE	21	0.4 %
17	CALL	332	6.5 %
18	CHBLT	1	0.0 %
19	CHKSTK	35	0.6 %

20	CLINVSBL	24	0.4 %
21	CNBEQ	79	1.5 %
22	CNBGE	32	0.6 %
23	CNBGT	123	2.4 %
24	CNBLE	14	0.2 %
25	CNBLT	263	5.1 %
26	CNBNE	77	1.5 %
27	CONS	3	0.0 %
28	DECR	12	0.2 %
29	DIFF	75	1.4 %
30	GFTYPE	39	0.7 %
31	HBMOVX	46	0.9 %
32	HBTEQ	6	0.1 %
33	HBXMOV	237	4.6 %
34	HGSIZE	17	0.3 %
35	HPMOVX	2	0.0 %
36	HPXMOV	24	0.4 %
37	HSOBJ	1	0.0 %
38	HSSIZE	1	0.0 %
39	INBF	2	0.0 %
40	INCR	146	2.8 %
41	INTEST	55	1.0 %
42	JCALL	15	0.2 %
43	JMP	16	0.3 %
44	LAND	7	0.1 %
45	MOV	1046	20.6 %
46	MOVNIL	42	0.8 %
47	NCONS	12	0.2 %
48	NXCONS	24	0.4 %
49	NXFLOAT	2	0.0 %
50	NXHB	1	0.0 %
51	NXNUMB	1	0.0 %
52	NXSTRG	5	0.0 %
53	NXSYMB	3	0.0 %
54	NXVECT	4	0.0 %
55	PLUS	75	1.4 %
56	POP	306	6.0 %
57	POPR	26	0.5 %
58	PUSH	364	7.1 %
59	REM	7	0.1 %
60	RETURN	383	7.5 %
61	SFTYPE	1	0.0 %
62	SOBGEZ	56	1.1 %
63	SPTYPE	1	0.0 %
64	STACK	15	0.2 %
65	TOPST	11	0.2 %
66	TOPSTR	9	0.1 %
67	XCONS	7	0.1 %
68	XTOPST	34	0.6 %
69	XTOPSTR	9	0.1 %

Occurrences dynamiques des instructions

Nombre d'instructions (en K instructions) : 5072

1	MOV	1046	20.6 %
2	RETURN	383	7.5 %
3	PUSH	364	7.1 %

4	CALL	332	6.5 %
5	POP	306	6.0 %
6	CNBLT	263	5.1 %
7	HBXMOV	237	4.6 %
8	BFNIL	228	4.4 %
9	INCR	146	2.8 %
10	BTNIL	137	2.7 %
11	CNBGT	123	2.4 %
12	BTCONS	111	2.1 %
13	BRX	103	2.0 %
14	BRA	92	1.8 %
15	CNBEQ	79	1.5 %
16	CNBNE	77	1.5 %
17	DIFF	75	1.4 %
18	PLUS	75	1.4 %
19	BFSYMB	67	1.3 %
20	SOBGEZ	56	1.1 %
21	INTEST	55	1.0 %
22	CABEQ	51	1.0 %
23	HEMOVX	46	0.9 %
24	BFCONS	42	0.8 %
25	MOVNIL	42	0.8 %
26	GFTYPE	39	0.7 %
27	CHKSTK	35	0.6 %
28	XTOPST	34	0.6 %
29	CNBGE	32	0.6 %
30	POPR	26	0.5 %
31	CLINVSBL	24	0.4 %
32	HPXMOV	24	0.4 %
33	NXCONS	24	0.4 %
34	BTSTRG	21	0.4 %
35	CABNE	21	0.4 %
36	BRI	20	0.3 %
37	HGSIZE	17	0.3 %
38	BTVAR	16	0.3 %
39	JMP	16	0.3 %
40	ADJSTK	15	0.2 %
41	JCALL	15	0.2 %
42	STACK	15	0.2 %
43	CNBLE	14	0.2 %
44	BFFIX	13	0.2 %
45	DECR	12	0.2 %
46	NCONS	12	0.2 %
47	TOPST	11	0.2 %
48	TOPSTR	9	0.1 %
49	XTOPSTR	9	0.1 %
50	LAND	7	0.1 %
51	REM	7	0.1 %
52	XCONS	7	0.1 %
53	HBTEQ	6	0.1 %
54	NXSTRG	5	0.0 %
55	BTFIX	4	0.0 %
56	NXVECT	4	0.0 %
57	BTSYMB	3	0.0 %
58	CONS	3	0.0 %
59	NXSYMB	3	0.0 %
60	HPMOVX	2	0.0 %
61	INBF	2	0.0 %
62	NXFLOAT	2	0.0 %
63	CHBLT	1	0.0 %
64	HSOBJ	1	0.0 %

65	HSSIZE	1	0.0 %
66	NXHB	1	0.0 %
67	NXNUMB	1	0.0 %
68	SFTYPE	1	0.0 %
69	SPTYPE	1	0.0 %

Occurrences dynamiques des opérandes

Nombre d'opérandes (en K opérandes) : 8526

1	A1	1138	13.3 %
2	A2	685	8.0 %
3	A3	1550	18.1 %
4	A4	926	10.8 %
5	VAL	15	0.1 %
6	CAR	148	1.7 %
7	CDR	160	1.8 %
8	CVAL	116	1.3 %
9	PLIST	3	0.0 %
10	FVAL	38	0.4 %
11	ALINK	22	0.2 %
12	OVAL	1	0.0 %
13	PKGC	12	0.1 %
14	PNAME	15	0.1 %
15	IMAT	109	1.2 %
16	IMAD	140	1.6 %
17	MEMADR	1099	12.8 %
18	NUMB	790	9.2 %
19	PCMOD	1559	18.2 %

Accès mémoire dynamiques

Nombre d'accès mémoire (en K accès) : 8546

1	CODE	5072	59.3 %
2	STACK	1469	17.1 %
3	MEMADR	1099	12.8 %
4	OBJ	571	6.6 %
5	OBJIND	335	3.9 %

22.2 Premier test de l'interpréteur

Ce test consiste à exécuter la fameuse fonction de Fibonacci qui est définie de la manière suivante :

```
(DE FIB (n)
  (COND ((EQ n 1) 1)
        ((EQ n 2) 1)
        (T (+ (FIB (1- n)) (FIB (- n 2))))))
```

Le calcul interprété est : (FIB 20)

Occurrences dynamiques des instructions (alpha)

Nombre d'instructions (en K instructions) : 3530

1	ADJSTK	13	0.3 %
2	BFCONS	128	3.6 %
3	BFFIX	33	0.9 %
4	BFNIL	174	4.9 %
5	BFSYMB	150	4.2 %
6	BRA	79	2.2 %
7	BRI	26	0.7 %
8	BRX	83	2.3 %
9	BTCONS	283	8.0 %
10	BTFLOAT	33	0.9 %
11	BTNIL	63	1.7 %
12	BTVAR	39	1.1 %
13	CABEQ	67	1.8 %
14	CABNE	39	1.1 %
15	CALL	117	3.3 %
16	CHKSTK	70	1.9 %
17	CNBEQ	6	0.1 %
18	CNBNE	6	0.1 %
19	DIFF	13	0.3 %
20	GFTYPE	70	1.9 %
21	INCR	26	0.7 %
22	INTEST	150	4.2 %
23	JCALL	13	0.3 %
24	JMP	26	0.7 %
25	MOV	780	22.0 %
26	MOVNIL	17	0.4 %
27	PLUS	19	0.5 %
28	POP	256	7.2 %
29	POPR	52	1.4 %
30	PUSH	326	9.2 %
31	RETURN	174	4.9 %
32	SOBGEZ	19	0.5 %
33	SOBGTZ	13	0.3 %
34	STACK	26	0.7 %
35	TOPST	26	0.7 %
36	TOPSTR	26	0.7 %
37	XTOPST	63	1.7 %
38	XTOPSTR	26	0.7 %

Occurrences dynamiques des instructions

Nombre d'instructions (en K instructions) : 3530

1	MOV	780	22.0 %
2	PUSH	326	9.2 %
3	BTCONS	283	8.0 %
4	POP	256	7.2 %
5	BFNIL	174	4.9 %
6	RETURN	174	4.9 %

7	BFSYMB	150	4.2 %
8	INTEST	150	4.2 %
9	BFCONS	128	3.6 %
10	CALL	117	3.3 %
11	BRX	83	2.3 %
12	BRA	79	2.2 %
13	CHKSTK	70	1.9 %
14	GFTYPE	70	1.9 %
15	CABEQ	67	1.8 %
16	BTNIL	63	1.7 %
17	XTOPST	63	1.7 %
18	POPR	52	1.4 %
19	BTVAR	39	1.1 %
20	CABNE	39	1.1 %
21	BFFIX	33	0.9 %
22	BTFLOAT	33	0.9 %
23	BRI	26	0.7 %
24	INCR	26	0.7 %
25	JMP	26	0.7 %
26	STACK	26	0.7 %
27	TOPST	26	0.7 %
28	TOPSTR	26	0.7 %
29	XTOPSTR	26	0.7 %
30	PLUS	19	0.5 %
31	SOBGEZ	19	0.5 %
32	MOVNIL	17	0.4 %
33	ADJSTK	13	0.3 %
34	DIFF	13	0.3 %
35	JCALL	13	0.3 %
36	SOBGTZ	13	0.3 %
37	CNBEQ	6	0.1 %
38	CNBNE	6	0.1 %

Occurrences dynamiques des opérandes

Nombre d'opérandes (en K opérandes) : 5021

1	A1	1268	25.2 %
2	A2	693	13.8 %
3	A3	463	9.2 %
4	A4	403	8.0 %
5	CAR	278	5.5 %
6	CDR	278	5.5 %
7	CVAL	83	1.6 %
8	FVAL	70	1.3 %
9	IMAT	63	1.2 %
10	IMAD	109	2.1 %
11	MEMADR	357	7.1 %
12	NUMB	112	2.2 %
13	PCMOD	844	16.8 %

Accès mémoire dynamiques

Nombre d'accès mémoire (en K accès) : 5693

1	CODE	3530	62.0 %
2	STACK	1027	18.0 %
3	MEMADR	357	6.2 %
4	OBJ	779	13.6 %

22.3 Vitesse d'une fonction compilée

Ce test va exécuter la même fonction compilée.

Occurrences dynamiques des instructions (alpha)

Nombre d'instructions (en K instructions) : 123

1	ADJSTK	13	10.5 %
2	BRA	6	4.8 %
3	CALL	13	10.5 %
4	CNBNE	23	18.6 %
5	DECR	6	4.8 %
6	DIFF	6	4.8 %
7	MOV	6	4.8 %
8	PLUS	6	4.8 %
9	POP	6	4.8 %
10	PUSH	19	15.4 %
11	RETURN	13	10.5 %
12	XSPMOV	6	4.8 %

Occurrences dynamiques des instructions

Nombre d'instructions (en K instructions) : 123

1	CNBNE	23	18.6 %
2	PUSH	19	15.4 %
3	ADJSTK	13	10.5 %
4	CALL	13	10.5 %
5	RETURN	13	10.5 %
6	BRA	6	4.8 %
7	DECR	6	4.8 %
8	DIFF	6	4.8 %
9	MOV	6	4.8 %
10	PLUS	6	4.8 %
11	POP	6	4.8 %
12	XSPMOV	6	4.8 %

Occurrences dynamiques des opérandes

Nombre d'opérandes (en K opérandes) : 197

1	A1	77	39.0 %
2	A2	13	6.5 %
3	NUMB	57	28.9 %
4	PCMOD	50	25.3 %

Accès mémoire dynamiques

Nombre d'accès mémoire (en K accès) : 180

1	CODE	123	68.3 %
2	STACK	57	31.6 %

22.4 Autre test de l'interpréteur

Ce test consiste à trier toute l'OBLIST au moyen de la fonction décrite dans la manuel (voir la fonction SORTL). La liste contient 2121 éléments.

Occurrences dynamiques des instructions (alpha)

Nombre d'instructions (en K instructions) : 5810

1	ADJSTK	4	0.0 %
2	BFCNS	132	2.2 %
3	BFFIX	4	0.0 %
4	BFNIL	205	3.5 %
5	BFSTRG	9	0.1 %
6	BFSYMB	188	3.2 %
7	BFVAR	15	0.2 %
8	BRA	47	0.8 %
9	BR1	13	0.2 %
10	BRX	113	1.9 %
11	BTCONS	347	5.9 %
12	BTFLOAT	1	0.0 %
13	BTNIL	105	1.8 %
14	BTSTRG	46	0.7 %
15	BTSYMB	19	0.3 %
16	BTVAR	21	0.3 %
17	CABEQ	77	1.3 %
18	CABNE	19	0.3 %
19	CALL	139	2.3 %
20	CHKSTK	101	1.7 %
21	CNBEQ	10	0.1 %
22	CNBGE	5	0.0 %
23	CNBGT	9	0.1 %
24	CNBLT	288	4.9 %
25	CNBNE	295	5.0 %
26	CONS	1	0.0 %
27	DECR	14	0.2 %
28	DIFF	1	0.0 %
29	GFTYPE	109	1.8 %
30	HBXMOV	576	9.9 %

31	HGSIZE	18	0.3 %
32	INCR	318	5.4 %
33	INTEST	194	3.3 %
34	JCALL	21	0.3 %
35	JMP	32	0.5 %
36	MOV	1042	17.9 %
37	MOVNIL	6	0.1 %
38	MOVXSP	9	0.1 %
39	POP	214	3.6 %
40	POPR	31	0.5 %
41	PUSH	329	5.6 %
42	QUO	1	0.0 %
43	RETURN	267	4.5 %
44	SOBGEZ	290	4.9 %
45	STACK	8	0.1 %
46	TOPST	25	0.4 %
47	TOPSTR	12	0.2 %
48	XSPMOV	9	0.1 %
49	XTOPST	59	1.0 %
50	XTOPSTR	12	0.2 %

Occurrences dynamiques des instructions

Nombre d'instructions (en K instructions) : 5810

1	MOV	1042	17.9 %
2	HBXMOV	576	9.9 %
3	BTCONS	347	5.9 %
4	PUSH	329	5.6 %
5	INCR	318	5.4 %
6	CNBNE	295	5.0 %
7	SOBGEZ	290	4.9 %
8	CNBLT	288	4.9 %
9	RETURN	267	4.5 %
10	POP	214	3.6 %
11	BFNIL	205	3.5 %
12	INTEST	194	3.3 %
13	BFSYMB	188	3.2 %
14	CALL	139	2.3 %
15	BFCONS	132	2.2 %
16	BRX	113	1.9 %
17	GFTYPE	109	1.8 %
18	BTNIL	105	1.8 %
19	CHKSTK	101	1.7 %
20	CABEQ	77	1.3 %
21	XTOPST	59	1.0 %
22	BRA	47	0.8 %
23	BTSTRG	46	0.7 %
24	JMP	32	0.5 %
25	POPR	31	0.5 %
26	TOPST	25	0.4 %
27	BTVAR	21	0.3 %
28	JCALL	21	0.3 %
29	BTSYMB	19	0.3 %
30	CABNE	19	0.3 %
31	HGSIZE	18	0.3 %
32	BFVAR	15	0.2 %
33	DECR	14	0.2 %

34	BRI	13	0.2 %
35	TOPSTR	12	0.2 %
36	XTOPSTR	12	0.2 %
37	CNBEQ	10	0.1 %
38	BFSTRG	9	0.1 %
39	CNBGT	9	0.1 %
40	MOVXSP	9	0.1 %
41	XSPMOV	9	0.1 %
42	STACK	8	0.1 %
43	MOVNIL	6	0.1 %
44	CNBGE	5	0.0 %
45	ADJSTK	4	0.0 %
46	BFFIX	4	0.0 %
47	BTFLOAT	1	0.0 %
48	CONS	1	0.0 %
49	DIFF	1	0.0 %
50	QUO	1	0.0 %

Occurrences dynamiques des opérandes

Nombre d'opérandes (en K opérandes) : 9667

1	A1	1867	19.3 %
2	A2	1176	12.1 %
3	A3	1319	13.6 %
4	A4	583	6.0 %
5	CAR	335	3.4 %
6	CDR	362	3.7 %
7	CVAL	124	1.2 %
8	FVAL	109	1.1 %
9	ALINK	20	0.2 %
10	PNAME	18	0.1 %
11	IMAT	94	0.9 %
12	IMAD	131	1.3 %
13	MEMADR	2143	22.1 %
14	NUMB	73	0.7 %
15	PCMOD	1313	13.5 %

Accès mémoire dynamiques

Nombre d'accès mémoire (en K accès) : 10714

1	CODE	5810	54.2 %
2	STACK	1090	10.1 %
3	MEMADR	2143	20.0 %
4	OBJ	1077	10.0 %
5	OBJIND	594	5.5 %

22.5 Test du Garbage Collector

Ce test est un appel du Garbage Collector.

Occurrences dynamiques des instructions (alpha)

Nombre d'instructions (en K instructions) : 401

1	BFFIX	25	6.2 %
2	BFMARK	2	0.4 %
3	BFNUMB	1	0.2 %
4	BRA	11	2.7 %
5	BTCONS	11	2.7 %
6	BTLISP	9	2.2 %
7	BTMARK	24	5.9 %
8	BTSTRG	6	1.4 %
9	BTSYMB	6	1.4 %
10	CABNE	2	0.4 %
11	CALL	15	3.7 %
12	CHBLT	1	0.2 %
13	CHKSTK	9	2.2 %
14	CLINVSBL	14	3.4 %
15	CNBLT	14	3.4 %
16	HGOBJ	1	0.2 %
17	HGSIZE	1	0.2 %
18	INCR	27	6.7 %
19	MCV	88	21.9 %
20	MOVNIL	14	3.4 %
21	NXCONS	24	5.9 %
22	NXFLOAT	2	0.4 %
23	NXHB	1	0.2 %
24	NXNUMB	1	0.2 %
25	NXSTRG	5	1.2 %
26	NXVECT	4	0.9 %
27	POP	9	2.2 %
28	PUSH	10	2.4 %
29	RETURN	15	3.7 %
30	STMARK	12	2.9 %
31	TCMARK	37	9.2 %

Occurrences dynamiques des instructions

Nombre d'instructions (en K instructions) : 401

1	MOV	88	21.9 %
2	TCMARK	37	9.2 %
3	INCR	27	6.7 %
4	BFFIX	25	6.2 %
5	BTMARK	24	5.9 %
6	NXCONS	24	5.9 %
7	CALL	15	3.7 %
8	RETURN	15	3.7 %

9	CLINVSBL	14	3.4 %
10	CNBLT	14	3.4 %
11	MOVNIL	14	3.4 %
12	STMARK	12	2.9 %
13	BRA	11	2.7 %
14	BTCONS	11	2.7 %
15	PUSH	10	2.4 %
16	BTLISP	9	2.2 %
17	CHKSTK	9	2.2 %
18	POP	9	2.2 %
19	BTSTRG	6	1.4 %
20	BTSYMB	6	1.4 %
21	NXSTRG	5	1.2 %
22	NXVECT	4	0.9 %
23	BFMARK	2	0.4 %
24	CABNE	2	0.4 %
25	NXFLOAT	2	0.4 %
26	BFNUMB	1	0.2 %
27	CHBLT	1	0.2 %
28	HGOBJ	1	0.2 %
29	HGSIZE	1	0.2 %
30	NXHB	1	0.2 %
31	NXNUMB	1	0.2 %

Occurrences dynamiques des opérandes

Nombre d'opérandes (en K opérandes) : 614

1	A1	195	31.7 %
2	A2	65	10.5 %
3	A3	4	0.6 %
4	A4	105	17.1 %
5	VAL	10	1.6 %
6	CAR	24	3.9 %
7	CDR	24	3.9 %
8	CVAL	1	0.1 %
9	PLIST	1	0.1 %
10	FVAL	1	0.1 %
11	ALINK	5	0.8 %
12	OVAL	1	0.1 %
13	PKGC	1	0.1 %
14	PNAME	3	0.4 %
15	IMAD	2	0.3 %
16	MEMADR	20	3.2 %
17	NUMB	14	2.2 %
18	PCMOD	138	22.4 %

Accès mémoire dynamiques

Nombre d'accès mémoire (en K accès) : 544

1	CODE	401	73.7 %
2	STACK	49	9.0 %
3	MEMADR	20	3.6 %

4	OBJ	71	13.0 %
5	OBJIND	3	0.5 %

22.6 Bibliographie

- [Chailloux 83] Chailloux Jérôme, *LE_LISP 80 version 12, le manuel de référence*, rapport technique INRIA no 27, Juillet 1983.
- [Chailloux&al 84] Chailloux Jérôme, Devin Matthieu et Hullot Jean-Marie, *LE_LISP : a Portable an Efficient LISP System* 1984 ACM Symposium on LISP and Functional Programming, Austin, Texas.
- [Chailloux 85] Chailloux Jérôme, *LE_LISP version 15 : le manuel de référence* INRIA, Février 1985.
- [Devin 85] Devin Matthieu, *Le portage du système LE_LISP : mode d'emploi*, rapport technique INRIA no 50, Mars 1985.
- [McCarthy 62] , *LISP 1.5 Programmer's manual*, the M.I.T. Press, Cambridge, Mass., 1962.

Index de la machine LLM3

ACOS op1,op2 [instruction]	16
ADJSTK op [instruction]	7
ADR adr [pseudo-instruction]	4
ADRHL op1,op2,op3 [instruction]	24
ALINK(accum) [opérande]	11
APFILE chan,strg,cc [instruction]	22
ASIN op1,op2 [instruction]	16
ATAN op1,op2 [instruction]	16
BCONS [variable globale]	8
BFCONS op,lab [instruction]	9
BFFIX op,lab [instruction]	13
BFFLOAT op,lab [instruction]	15
BFINVSBL accum,lab [instruction]	10
BFLOAT [variable globale]	15
BFMARK accum,lab [instruction]	8
BFNIL op,lab [macro LLM3]	10
BFNUMB op,lab [instruction]	13
BFSTRG op,lab [instruction]	18
BFSYMB op,lab [instruction]	11
BFVAR op,lab [instruction]	12
BFVECT op,lab [instruction]	17
BHEAP [variable globale]	19
BNUMB [variable globale]	13
BRA lab [instruction]	5
BRI op [instruction]	5
BRX adr,op [instruction]	5
BSTACK [variable globale]	6
BSTRG [variable globale]	18
BSYMB [variable globale]	10
BTCONS op,lab [instruction]	9
BTFIX op,lab [instruction]	13
BTFLOAT op,lab [instruction]	15
BTINVSBL accum,lab [instruction]	10
BTLISP op,lab [instruction]	8
BTMARK accum,lab [instruction]	8
BTNIL op,lab [macro LLM3]	10
BTNUMB op,lab [instruction]	13
BTSTRG op,lab [instruction]	18
BTSYMB op,lab [instruction]	11
BTVAR op,lab [instruction]	12
BTVECT op,lab [instruction]	17
BVAR [variable globale]	12
BVECT [variable globale]	17
CABEQ op1,op2,lab [instruction]	5
CABNE op1,op2,lab [instruction]	5
CALL lab [instruction]	6
CALLG op1,op2 [instruction]	23
CALLI op [instruction]	6
CAR(accum) [opérande]	9
CCONS [variable globale]	8
CDR(accum) [opérande]	9
CFBEQ op1,op2,lab [instruction]	16
CFBGE op1,op2,lab [instruction]	16
CFBGT op1,op2,lab [instruction]	16
CFBLE op1,op2,lab [instruction]	16
CFBLT op1,op2,lab [instruction]	16
CFBNE op1,op2,lab [instruction]	16

Index de la machine LLM3

CFLOAT [variable globale]	15
CHBLT op1,op2,lab [instruction]	19
CHEAP [variable globale]	19
CHKSTK op,lab [instruction]	6
CLINE strg [instruction]	23
CLINVSBL accu [instruction]	10
CNBEQ op1,op2,lab [instruction]	14
CNBGE op1,op2,lab [instruction]	14
CNBGT op1,op2,lab [instruction]	14
CNBLE op1,op2,lab [instruction]	14
CNBLT op1,op2,lab [instruction]	14
CNBNE op1,op2,lab [instruction]	14
CONS op1,op2 [instruction]	9
CONVTOK op1,op2,op3 [instruction]	8
COREST strg,cc [instruction]	23
CORSAV strg,cc [instruction]	22
COS op1,op2 [instruction]	16
CSTRG [variable globale]	18
CSYMB [variable globale]	10
CSYMB [variable globale]	13
CVAL(accu) [opérande]	11
CVATOF strg,n,op,lab [instruction]	15
CVECT [variable globale]	17
CVFTOA op1,strg,op2 [instruction]	15
DECR op[,lab] [instruction]	13
DIFF op1,op2[,lab] [instruction]	13
ECOMP op1,op2,lab1,lab2,lab3 [instruction]	16
EDIVIDE op1,op2,op3,op4,op5 [instruction]	16
END [pseudo-instruction]	4
EPLUS op1,op2,op3,op4,op5 [instruction]	16
ESTACK [variable globale]	6
ETIMES op1,op2,op3,op4,op5,op6 [instruction]	16
EXP op1,op2 [instruction]	16
FCLOS chan,cc [instruction]	22
FDELE strg,cc [instruction]	22
FDIFF op1,op2,lab [instruction]	15
FENTRY nom,type [pseudo-instruction]	4
FIX op [instruction]	15
FLOAT op [instruction]	15
FPLUS op1,op2,lab [instruction]	15
FQUO op1,op2,lab [instruction]	16
FRENA strg1,strg2,cc [instruction]	22
FTIMES op1,op2,lab [instruction]	16
FVAL(accu) [opérande]	11
GETENVRN strg1,strg2,op [instruction]	23
GETGLOBAL strg,op [instruction]	23
GFTYPE op1,op2 [instruction]	11
GPTYPE op1,op2 [instruction]	11
HASHTAB [variable]	26
HBLT heap1,heap2,heap3 [instruction]	20
HBMVM n1,strg1,n2,strg2,n3 [instruction]	20
HBMVM op,strg,index [instruction]	20
HBTEQ n1,strg1,n2,strg2,n3,lab [instruction]	20
HBXMOV strg,index,op [instruction]	20
HGOBJ op1,op2 [instruction]	21
HGSIZE op1,op2 [instruction]	20
HLADR op1,op2,op3 [instruction]	24
HPMVM n1,vect1,n2,vect2,n3 [instruction]	20
HPMVM op,vect,n [instruction]	20
HPXMOV vect,n,op [instruction]	20
HSOBJ op1,op2 [instruction]	21

Index de la machine LLM3

HSSIZE op1,op2 [instruction]	21
IMPURE [pseudo-instruction]	4
INBF chan,strg,siz,cc [instruction]	22
INCR op[,lab] [instruction]	13
INFILE chan,strg,cc [instruction]	22
INTEST [instruction]	23
INTOFF [instruction]	23
INTON [instruction]	23
JCALL lab [instruction]	6
JMP lab [instruction]	5
LABEL [pseudo-instruction]	4
LAND op1,op2 [instruction]	14
LOG op1,op2 [instruction]	16
LOG10 op1,op2 [instruction]	16
LOR op1,op2 [instruction]	14
LSHIFT op1,op2 [instruction]	14
LXOR op1,op2 [instruction]	14
MAKCST lab,plen,pname [instruction]	12
MAKFNT lab,plen,pname [instruction]	12
MAXCHAN [variable globale]	21
MEMGET op1,op2 [instruction]	24
MEMSET op1,op2 [instruction]	24
MOV op1,op2 [instruction]	5
MOVBM size,asource,strg	20
MOVNIL op [macro LLM3]	10
MOVXSP op1,op2 [instruction]	7
MSTACK [variable globale]	6
NCONS op [macro LLM3]	9
NEGATE op [instruction]	13
NXCONS op,lab [instruction]	8
NXFLOAT op,lab [instruction]	15
NXHB size,heap [instruction]	19
NXHF heap [instruction]	19
NXHP size,heap [instruction]	19
NXNUMB op,lab [instruction]	13
NXSTRG op,lab [instruction]	18
NXSYMB op,lab [instruction]	11
NXVECT op,lab [instruction]	17
OUFIL chan,strg,cc [instruction]	22
OUTF chan,strg,size,cc [instruction]	22
OUTFL chan,strg,size,cc [instruction]	22
OVAL(accum) [opérande]	11
PKGC(accum) [opérande]	11
PLIST(accum) [opérande]	11
PLUS op1,op2[,lab] [instruction]	13
PNAME(accum) [opérande]	11
POP op [instruction]	7
POPR accum,op [instruction]	7
POWER op1,op2,op3 [instruction]	16
PURE [pseudo-instruction]	4
PUSH op [instruction]	7
QUO op1,op2 [instruction]	14
REM op1,op2 [instruction]	14
RETURN [instruction]	7
RUNTIME op [instruction]	23
SETBVAR op [instruction]	12
SFREEL op [macro LLM3]	9
SFTYPE op1,op2 [instruction]	11
SIN op1,op2 [instruction]	16
SLEEP op [instruction]	23
SOBGEZ op,lab [instruction]	5

Index de la machine LLM3

SOBGZ op,lab [instruction]	5
SPTYPE op1,op2 [instruction]	11
SQRT op1,op2 [instruction]	16
SSTACK op [instruction]	6
STACK op [instruction]	6
STINVSBL accu [instruction]	10
STMARK accu [instruction]	8
TCMARK accu,lab [instruction]	8
TIMES op1,op2[,lab] [instruction]	13
TITLE nom [pseudo-instruction]	4
TOPST op [instruction]	7
TOPSTR accu,op [instruction]	7
TTYCRLF [instruction]	21
TTYIN op [instruction]	21
TTYIS op,cc [instruction]	21
TTYMSG n,<suite-de-caractères> [instruction]	21
TTYSTRG n,strg [instruction]	21
XCONS op1,op2 [instruction]	9
XDEFI nom [pseudo-instruction]	4
XDEFP nom [pseudo-instruction]	4
XREFI module,nom [pseudo-instruction]	4
XREFP module,nom [pseudo-instruction]	4
XSPMOV op1,op2 [instruction]	7
XTOPST op [instruction]	7
XTOPSTR accu,op [instruction]	7

Table des matières

1 Introduction.....	1
2 Les notions de base LLM3.....	2
2.1 Les objets LLM3.....	2
2.2 Les modules LLM3.....	2
2.3 Les macros LLM3.....	2
2.4 Les variables globales et le lanceur.....	2
3 Le Format des instructions LLM3.....	2
4 Les opérandes des instructions LLM3.....	3
4.1 Les registres rapides.....	3
4.2 Les opérandes immédiats.....	3
4.3 Les opérandes qui accèdent à la mémoire.....	3
5 Les Pseudos-Instructions.....	4
6 Les Instructions de base.....	5
6.1 Les transferts de pointeurs.....	5
6.2 Les comparaisons de pointeurs.....	5
6.3 Le contrôle.....	5
7 La Pile.....	6
7.1 Organisation mémoire.....	6
7.2 Gestion du pointeur de pile.....	6
7.3 Pile de contrôle.....	6
7.4 Pile de données.....	7
8 Le garbage-collector (GC).....	8
9 Les cellules de liste (CONS).....	8
9.1 Organisation mémoire.....	8
9.2 Test du type cellule de liste.....	9
9.3 Accès aux champs des cellules de liste.....	9
9.4 Création (allocation) d'une cellule de liste.....	9
9.5 Bit invisible.....	9
10 NIL.....	10
11 Les symboles.....	10
11.1 Organisation mémoire.....	10
11.2 Test de type symbole.....	11
11.3 Accès aux différents champs d'un symbole.....	11
11.4 Instructions spécialisées.....	11
11.5 Création statique.....	12
11.6 Les variables.....	12
12 Les nombres.....	12
12.1 Les nombres entiers sur 16 bits.....	12
12.1.1 L'organisation mémoire.....	12
12.1.2 Les tests de type.....	13
12.1.3 Les instructions de calcul.....	13
12.1.4 Les comparaisons numériques entières.....	14
12.1.5 Les instructions logiques.....	14
12.2 Les nombres flottants.....	14
12.2.1 Les tests de type nombre flottant.....	15
12.2.2 Conversions.....	15
12.2.3 Les instructions de bases.....	15
12.2.4 Les comparaisons.....	16
12.2.5 Les fonctions circulaires et mathématiques.....	16
12.3 Les nombres à précision variable.....	16
13 Les Vecteurs de Pointeurs LISP.....	17
13.1 Organisation mémoire.....	17
13.2 Test de type vecteur de pointeur.....	17
13.3 Allocation.....	17
13.4 Accès aux éléments d'un vecteur de pointeurs.....	17
14 Les Chaînes de Caractères.....	18
14.1 Organisation mémoire.....	18
14.2 Test de type chaîne de caractères.....	18

Table des matières

14.3 Allocation	18
14.4 Accès aux caractères	18
15 Zone du tas (HEAP)	19
15.1 Organisation	19
15.2 Autres instructions	19
15.3 La récupération de l'espace	19
15.4 Accès indexé	20
15.4.1 de type octet	20
15.4.2 de type pointeurs	20
15.5 Accès aux champs cachés	20
16 Les Entrées/Sorties	21
16.1 Les instructions sur le canal terminal	21
16.2 Les instructions sur les fichiers	21
16.3 Les instructions sur les images mémoire	22
17 Les instructions système	23
18 Les instructions d'accès à la mémoire	24
19 Les Fonctions	24
19.1 Les types des fonctions	24
19.2 Règle d'appel des fonctions	24
20 Programmer en LLM3	25
20.1 Exemple	25
20.2 traitement des SUBRN	25
20.3 L'oblist (la liste des symboles)	26
21 Occurrences Statiques du noyau de l'interprète	27
22 Occurrences Dynamiques	33
22.1 Le chargement de l'environnement initial	34
22.2 Premier test de l'interpréteur	37
22.3 Vitesse d'une fonction compilée	40
22.4 Autre test de l'interpréteur	41
22.5 Test du Garbage Collector	44
22.6 Bibliographie	47

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique